Introduction to Geoprocessing Scripts Using Python®

Contents

Introduction

The ArcGIS 10 geoprocessing framework provides access to automation of geoprocessing functionality through the Python open-source scripting language.

This course introduces Python's language syntax and modules, which can be used to access and automate the geoprocessing functionality through Python scripts. These scripts can be run in the Python development environment, within the Python window in ArcMap and ArcCatalog, and as custom script tools within ArcToolbox.

The course also introduces ArcPy, the new Esri-developed Python site package that integrates Python scripts into ArcGIS Desktop 10. With ArcPy, your Python scripts can now access additional functionality to work with your maps and data beyond the geoprocessing framework.

Course goals

By the end of this course, you will be able to:

- Understand the basics of Python language syntax
- Work with the ArcPy site package in ArcGIS Desktop and PythonWin
- Incorporate cursor objects, describe objects, and Python lists into scripts
- Automate the production, printing, and exporting of a map series using ArcPy
- Create new script tools in ArcToolbox and access resources for debugging Python code
- Understand commonly used ArcPy classes and functions
- Create geometry objects for use with geoprocessing tools and updating features
- Change table and feature schema for migrating data to new feature classes and tables

Using the course workbook

The course workbook is an integral part of your learning experience. During class, you will use the workbook to complete activities and exercises that reinforce specific tasks and skills. After class, the workbook is your personal reference to review activities or work through exercises again to reinforce what you've learned.

Essential elements include:

- **Lessons**—learning objectives at the beginning of each lesson to help you find the information you're looking for
- **Guided activities**—interactive activities to reinforce key topics
- **Exercises**—step-by-step instructions for accomplishing essential tasks and building skills
- **Review**—questions and answers that reinforce key concepts
- **Appendixes**—your guide to additional resources
- **CD**—data necessary for completing the course exercises

Additional resources

Refer to the following resources to learn more:

ArcGIS Resource Center

http://resources.arcgis.com/content/geoprocessing
This site provides unified access to Web-based Help, online content, and technical support for geoprocessing tools and tasks.

Python Programming Language -- Official Website

http://www.python.org
This site provides access to Python tutorials, documentation, and forums.

Installing the course data

To complete the workbook exercises for this course, you must install the necessary data. The data is stored on a disc and will be copied to your hard drive by an automated install program.

- ❑ Remove the disc from the back of your course workbook and place it in the disk drive.

- ❑ In the installation wizard, do the following:
  - Click Yes to accept the license agreement.
  - Click Next on the welcome panel.
    By default, the course data will be installed to a **C:\Student** folder.

> **Note:** If you need to install the course data to a different location, browse to and select the folder, then click OK. Note the location of the folder so that you can easily access the data in the exercises.

  - Click Next.
  - Click Finish when the installation is complete.
- ❑ Remove the disc and return it to its sleeve in your workbook.

1 What is Python?

Introduction

This lesson will introduce you to Python, a free open-source, cross-platform programming language. Programs written in the Python scripting language are called scripts, which are not compiled but rather interpreted when you run them. Python is automatically installed when you install ArcGIS. You will use PythonWin 2.6 to enter, debug, and test your code. PythonWin is not installed by ArcGIS, but is a free download available on the Internet.

All scripting languages have a standard set of functionality. For example, every scripting language has the ability to comment code, execute loops, concatenate strings, run decision-making statements, and execute a set of built-in functions. The main differences between scripting languages are the syntax and the type of built-in functionality.

Learning objectives

After completing this lesson, you will be able to:

- Write scripts with correct Python syntax
- Choose where to write Python scripts
- Locate Python resources

What are some reasons why you might want to write a script?

Notes

Integrated Development Environments (IDEs)

When you are ready to write your first Python script, there are many different applications from which you can choose. A Python script is simply a text file with a .py extension, so you can use Notepad, WordPad, or another text editor. These applications allow you to quickly write your Python code, but there is no way to test, run, or debug your code. Writing code at the Python command prompt is also acceptable, but the command prompt does not provide any debugging tools.

An IDE provides an environment to write, run, and debug code from one location.

PythonWin
- Windows only
- Built-in debugger

IDLE
- Unix
- Linux
- Windows

Others
- Wing
- PyDev
- Boa Constructor

In course exercises, you will use PythonWin, which has already been installed for you.

PythonWin	IDLE
Python IDE and GUI Framework for Windows	Python's Integrated DeveLopment Environment
Download from **http://sourceforge.net/projects/pywin32**	Automatically installed with ArcGIS
Microsoft Windows only ▪ Windows-based UI with menus and toolbars ▪ Windows contained within a single application	Most platforms that support Python ▪ Menu-driven UI ▪ Many windows

PythonWin and IDLE each provide two types of windows:

- The script window in which you can save all your code to a .py file, re-open the file at any time, and distribute the file to other users.
- The Interactive Window (PythonWin) or the Python Shell window (IDLE) in which you can evaluate one expression at a time, view error messages generated from the script window, and report output from print statements.

What are variables and why would you use them in a Python script?

Notes

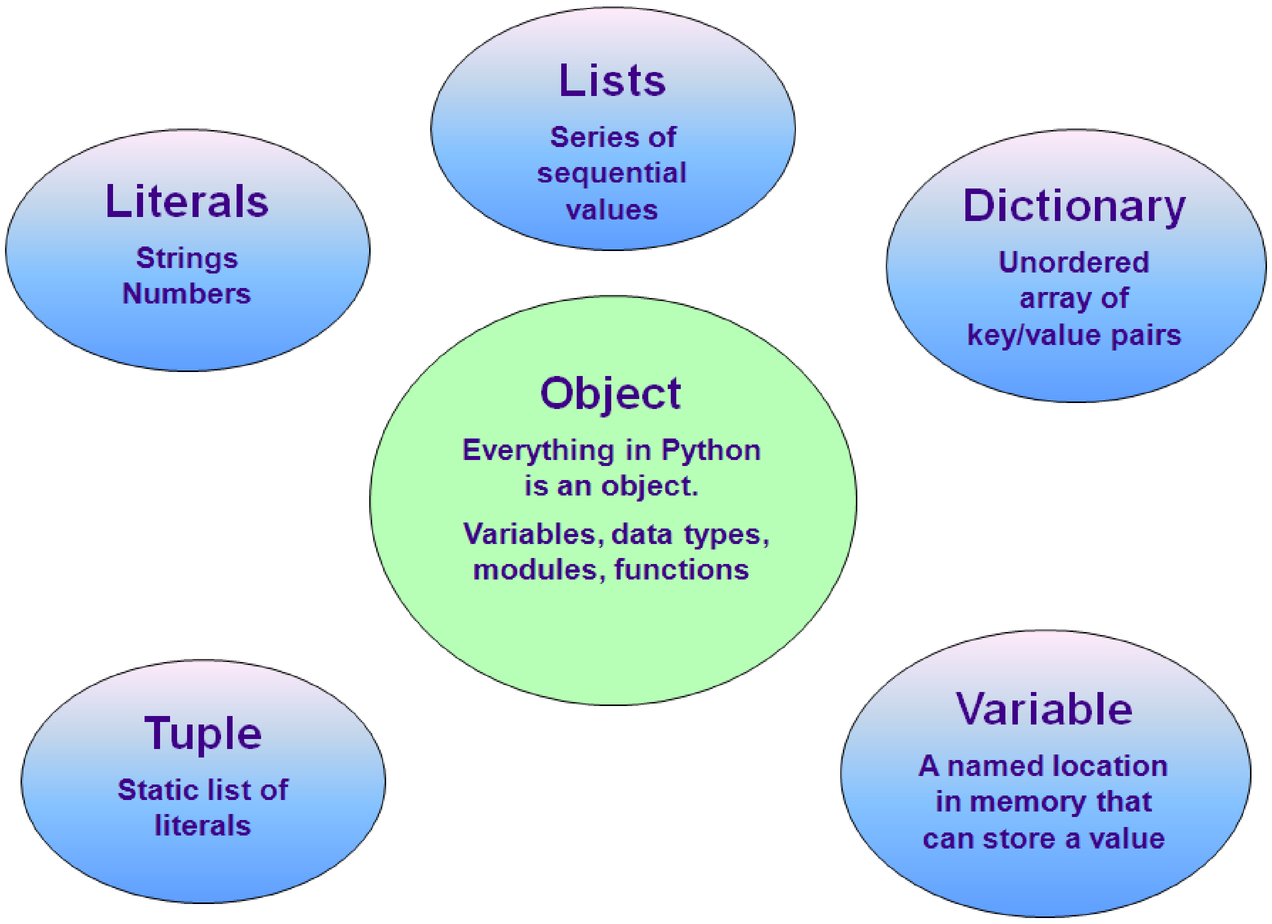
Python terminology

Throughout this lesson, you will explore Python data types, Python built-in functions, modules available to import into your scripts, as well as how to write conditional statements and work with Python lists and dictionaries.

Variable	A temporary container for data in memory.
Python data type	The types of data that are native to Python.
Python function	Something that Python knows how to do, such as open a file or print a value. Functions typically return a value that can be stored in a variable.
Python module	Most of Python's functions are not automatically available to you, you need to import the module into your code. Once imported, the module will give you access to its functions.
Statement	Performs an action, much like a function.
Conditional statement	A Python statement that will execute code if it is True.
Python list	A series of sequential values.
Python dictionary	An unordered collection of values that contain a key and value pair. The value can be looked up in the dictionary by specifying the key.
Arguments	Values passed to the script at run time.

Python data types

In Python, there are several different data types you can use to store your values. The Literal data types store the most simple types of values, such as numbers and strings. Python Lists and Dictionaries are more complex, and are used to store data elements. The Tuple stores a series of literal values, which cannot be changed. The variable is simply a storage mechanism which has a name and stores the value assigned.


Lists
Series of sequential values
Literals
Strings
Numbers
Dictionary
Unordered array of key/value pairs
Object
Everything in Python is an object.
Variables, data types, modules, functions
Tuple
Static list of literals
Variable
A named location in memory that can store a value

Built-in functions

Python has many built-in functions. A function always returns a value.

Built-in means that Python automatically makes these functions available. A function is just a packaged block of code.

You do not need to import the functionality like you do for most functions. You can execute the block of code by calling the function by name.

Examples:

```
fc = "Railroads.shp"
len(fc)
---> 13

fields = ["OID", "Shape", "Name"]
len(fields)
---> 3

xExtent = (6260474.996464, 6338807.996464)
max(xExtent)
---> 6338807.996464

coord = file("C:\\XY.txt", "r").read()
---> # holds the contents of the XY.txt file

yCoord = 1811884.623964
round(yCoord)
---> 1811885.0
```

Python modules

Most of Python's functions are not built-in, but are stored in modules. To gain access to a particular function, you need to import the associated module into your script.

There are dozens of modules that collectively contain hundreds of functions. You can access a list of modules through Python's help system.

math
os
os.path
time
random
keyword
shutil
sys

To import a module into your script, use the `import` statement, followed by the name of the module. After the module is imported, you can call any function from the module by prefixing the name of the module to the name of the function. The syntax is:
`<modulename>.<functionname>`

Examples:

```
import math
math.sqrt(64) ---> 8.0
math.pow(10,2) ---> 100.0

import string
string.split("-155.3 -43.5") ---> ['-155.3', '-43.5']
string.upper("c:\\student") ---> 'C:\\STUDENT'

import os.path
os.path.basename("C:\\Student\\Streets.shp") ---> 'Streets.shp'
os.path.dirname("C:\\Student\\Streets.shp") ---> 'C:\\Student'
```

Statements

Python has many statements. A statement is like a function in that it performs an action; however, statements do not return values.

Statements are available directly from Python; they do not reside in modules.

Keywords
import
print
if..elif..else
try..except
del
from..import
return

Decision-making statements usually start by testing if a condition is true.

If the condition is true, then a series of lines gets executed. If the condition is false, then other code gets executed. If no condition is true, usually another block of code gets executed.

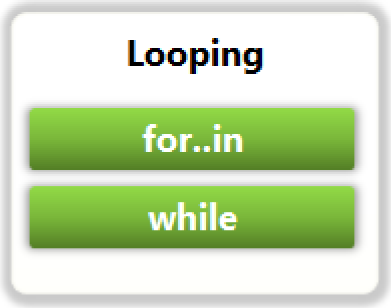
Example decisions that you might need to make in your code:

- If the variable holds a polygon feature class, calculate the area.
- If the feature dataset does not have a topology, create one.
- If the user did not browse to a shapefile, then report back an error.

Note: When testing to see if a particular condition it true, always use two equal signs (==), not one. Two equal signs are used to test for a condition, whereas one equal sign is used to assign a value to a variable. For example:

```
x = 1 # assignment
y = 8 + 2 # assignment
if x == 6: # testing a condition
```

Loops allow you to perform some action over and over again, slightly changing the input and output values.



Python has three types of loops.

- *While loops* keep executing over and over again while the condition is true. When the condition is false, the loop will stop.
- *Counted loops* increment and test a variable on each iteration of the loop. The last value is not executed. For example:

  ```
  for x in range(1,5) --> 1,2,3,4  # not 5
  ```

- *List loops* iterate over each value in a list. In other words, the loop will execute once for each value in the list. At the end of each `while` or `for` statement, a colon is required. If you forget the colon, you will get a syntax error. To finish a loop, dedent the code.

Tips

Python is mostly case sensitive.

- **All of Python's functions and statements are case sensitive.** You can tell if you have written a function or statement correctly in a Python script because it will turn blue. If a function or statement is not colored blue, then you have made an error, either in the spelling or in the case sensitivity.
- **Variable names are also case sensitive.** This is unlike many other programming and scripting languages. (Be careful when programming in Python if you are used to programming in VBScript or in Visual Studio .NET.) A special naming rule in Python applies to variables. Any variable name you use in Python must start with a character or underscore. Variable names cannot start with a number.

Note: At ArcGIS 10, all the geoprocessing properties and methods are case sensitive. When migrating your existing scripts from 9.3 to 10, make sure that you correct your geoprocessing properties or methods.

- **Pathnames are not case sensitive.** This means that any pathname can contain mixed case letters.

PythonWin keyboard shortcuts

PythonWin offers many keyboard shortcuts.
You can obtain a list by doing the following:

1. In PythonWin, on the Standard toolbar, click the Help Index button .
2. In the Select Help file dialog box, choose Pythonwin Reference and click OK.
3. In the PyWin32 window, on the Contents tab, navigate to:
    - Python for Win32 Extensions Help >
    - Pythonwin and win32ui >
    - Overviews >
    - Keyboard Bindings

Example:
You can type the first few characters of any variable name, then press **Alt+/** and Python will complete the variable name for you. You can also use **Ctrl+Spacebar** for auto-completion of a variable name.

Exercise 1: Learn the basics of Python

Estimated time: 30 minutes

In this exercise, you will become familiar with the Python scripting language, specifically comments, variables, built-in functions, modules, concatenation, decision-making statements, and loops. You will also become familiar with Python language best practices.

In this exercise, you will:

- Create a script and comment code
- Work with variables
- Work with built-in functions
- Work with modules
- Make decisions with statements
- Work with loops

Step 1: Create a script and comment code

In this step, you will open a new Python script and add some general comments to the top of it.

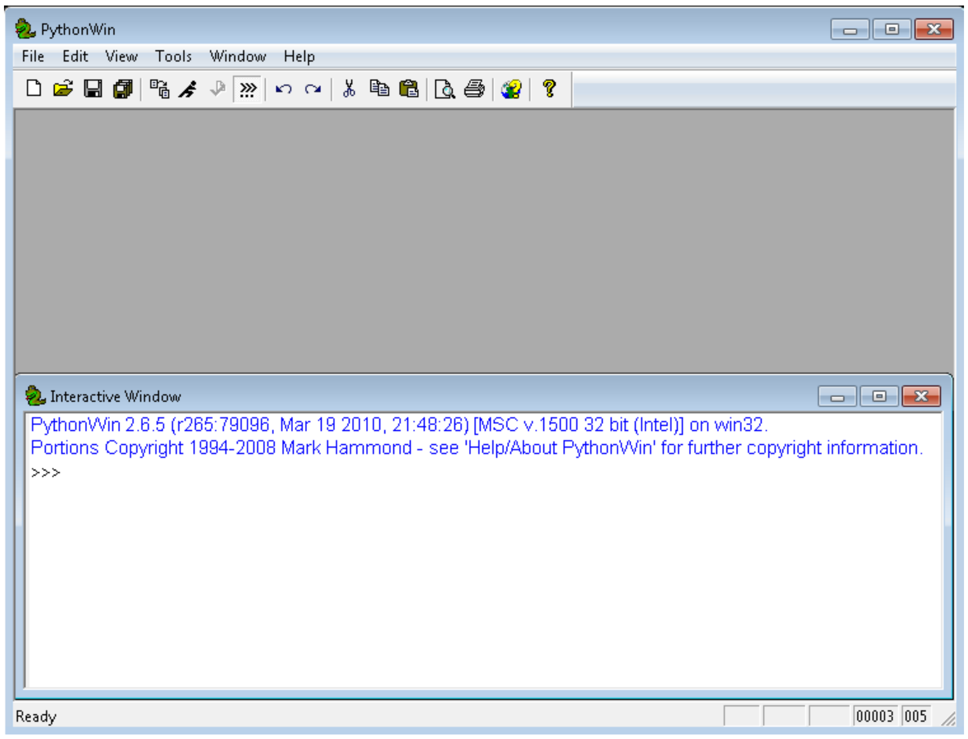
- ❑ Start PythonWin from the shortcut on your desktop or from the taskbar.

> **Note:** PythonWin can also be started from
> C:\Python26\ArcGIS10.0\Lib\site-packages\pythonwin\Pythonwin.exe

PythonWin is one of the integrated development environments (IDE) that you can use for writing Python scripts. When you open PythonWin, it opens with the Interactive Window. The Interactive Window has many purposes:

1. It provides a quick way to execute and test individual lines of code without saving that code.
2. It outputs error messages from scripts.
3. It shows the output from print statements.

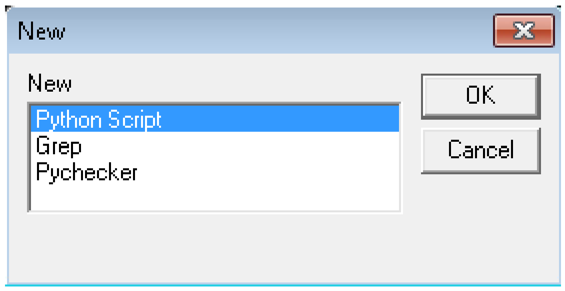
❑ Resize the Interactive Window so that it covers the bottom half of the application.



```
PythonWin 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>>
```

❑ From the File menu, choose New.

❑ In the New dialog box, verify that Python Script is selected.

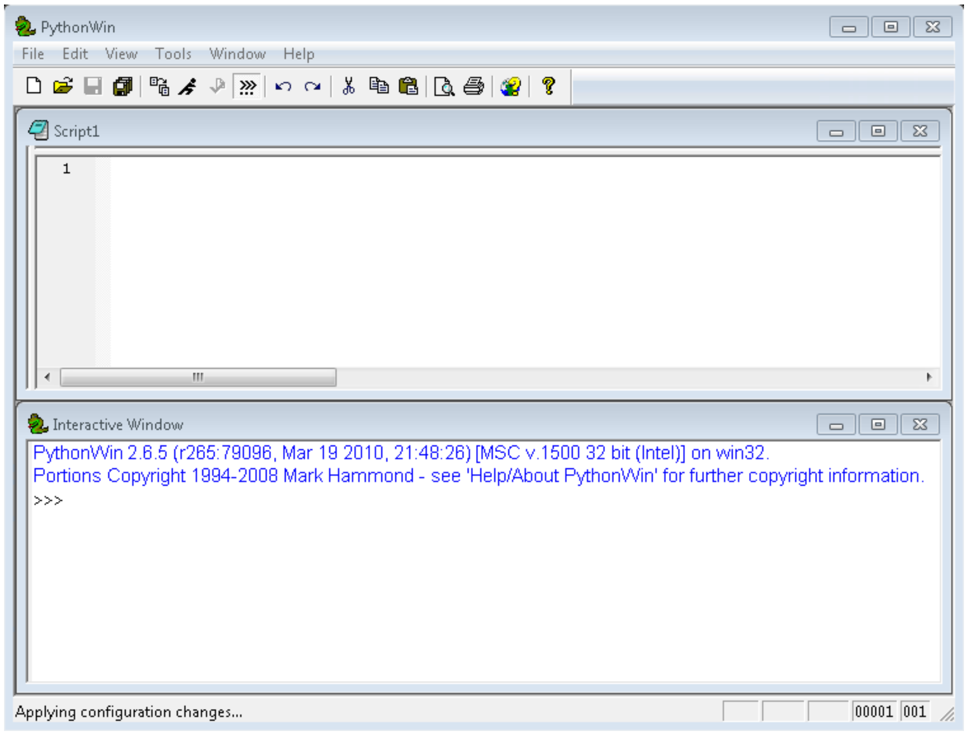


Note: You can run *Pychecker* against your Python script to look for a range of Python syntax errors and formatting. You can use *Grep* to scan a directory and find words within all types of files within a directory. For example: find "mdb" in all .py files.

❑ Click OK.

Currently, you have two windows within your application: the Interactive Window and the Script1 window. You can write lines of code in either window. However, the code that you write in the Interactive Window cannot be saved directly to a Python script. For this reason, most of your code will be written in a script window.

❑ Resize the Script1 window so that it covers the top half of the application.



```
PythonWin 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>>
```

Applying configuration changes...

❑ Click within the Script1 window, then click File > Save As.

❑ In the Save As dialog box, navigate to the C:\Student\PYTH\Exercise01 folder, and save the file as **BasicsOfPython.py**.

The first lines of code in any script should always be comments—lines of unexecutable code that are used to document a script. Include information about the author, date, and purpose of the script. You should also include comments throughout the code to explain individual lines or blocks of code. Properly commented code is essential to making a script comprehensible and readable to other programmers.

In Python, any line of code preceded by one or more pound signs (#) is interpreted as a comment. When you run a script, Python ignores all commented lines of code and only executes the uncommented lines of code.

- ❑ In the BasicsOfPython.py window, type the following.

```
# Name: <your name>
# Date: <current date>
# Purpose: This script is an exercise to learn the Python scripting language.
```

By default, commented code is green and italicized.

Step 2: Work with variables

Every scripting language, including Python, works with variables. You can think of a variable as a temporary container for information in a program. Throughout the execution of a program, the values in a variable can change.

In Python, variables do not need to be declared. In other words, you do not need to tell Python that a word or series of characters is going to represent a variable. All you need to do is assign a value to that word or series of characters. You do not need to tell Python what type of data a variable will hold. Python can determine the type of data that a variable will hold.

Variables can store many different types of data, including strings, numbers, lists, tuples, dictionaries, files, and so forth. In this step, you will use variables to store numbers, strings, and lists.

First, you will store a numeric value in a variable.

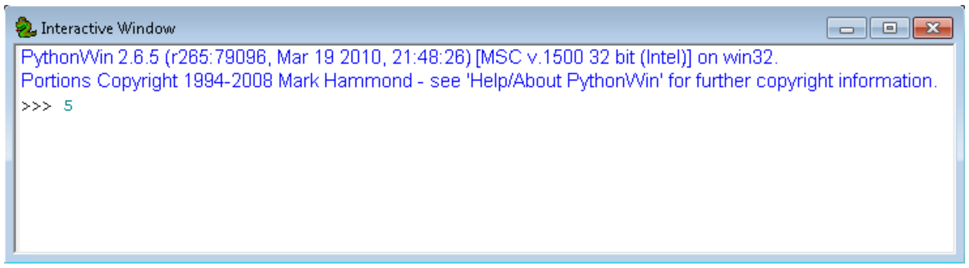
- ❑ In the BasicsOfPython.py window, below the comments, type the following code:

```
a = 5
print a
```

- ❑ On the Standard toolbar, click the Check button ✓ to check your code syntax.

The status bar reports that the script was checked successfully.

- ❑ Run the script:
  - On the Standard toolbar, click the Run button.
  - In the Run Script dialog box, click OK.



```
PythonWin 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>> 5
```

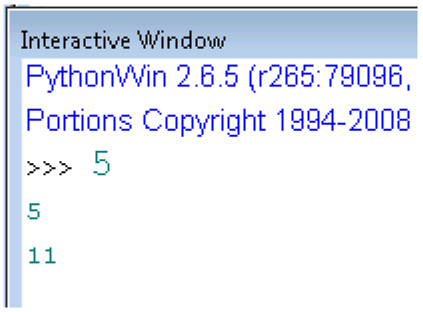
The value that is stored in `a` is printed to the Interactive Window. The print statement always sends output to the Interactive Window.

Variables can also store the result of an expression.

- ❑ In the BasicsOfPython.py window, after the `print a` statement, type the following code:

```
b = 5 + 6
print b
```

- ❑ Run the script. (Click the Run button. In the Run Script dialog box, click OK.)



```
PythonWin 2.6.5 (r265:79096,
Portions Copyright 1994-2008
>>> 5
5
11
```

The values that are stored in `a` and `b` are printed to the Interactive Window. The value of `b` holds the number `11`, which is the sum of `5 + 6`.

Variables can store string values. String values must be enclosed in single or double quotes.

> **Note:** When using the Interactive Window, there are several shortcut keys available.
> - Recall previous command: Ctrl+Up
> - Recall next command: Ctrl+Down
> - Auto-complete variable name: Ctrl+Space

❑ In the BasicsOfPython.py window, after the `print b` statement, type the following code:

```
c = "Hello"
print c
```

❑ Run the script.

The word `Hello` is the last line printed to the Interactive Window.

In addition to storing one value inside a variable, you can store multiple values inside a variable. These are known as lists. You can extract a single value (or a subset of values) from a list based on the index position.

❑ In the BasicsOfPython.py window, after the `print c` statement, type the following code:

```
d = ["Jack", "Diane", "Lisa"]
print d[0]
print d[2]
print d[-1]
```

❑ Run the script.

The last three lines of output in the Interactive Window are `Jack`, `Lisa`, and `Lisa`.

- `d[0]` returns `Jack`, which is the first value in the list (lists are zero-based).
- `d[2]` returns `Lisa`, which is the third value in the list.
- `d[-1]` returns `Lisa`, which is the first value from the right.

You can also use the same indexing functionality on an individual string. The indexing functionality is not limited to lists.

❑ In the BasicsOfPython.py window, after the `print d[-1]` statement, type the following code:

```
e = "Railroads.shp"
print e[3]
print e[-2]
print e[0:-4]
```

❑ Run the script.

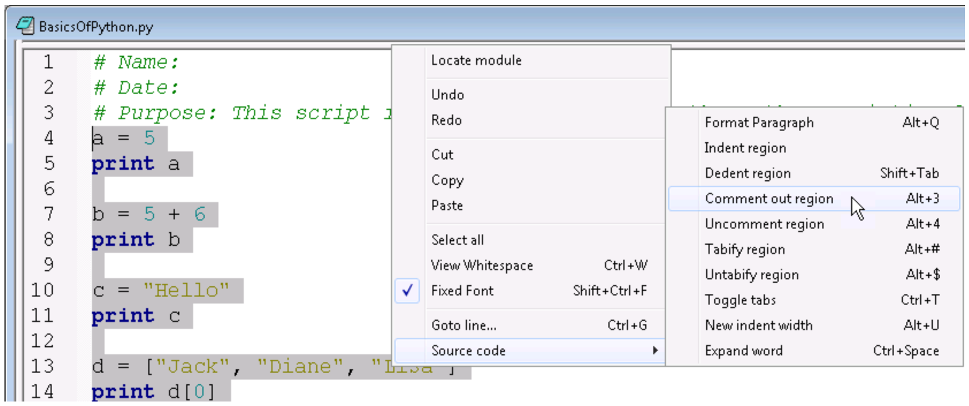
The last three lines of output to the Interactive Window are `l`, `h`, and `Railroads`.

- `e[3]` returns `l`, which is the fourth character in the string (strings are zero-based).
- `e[-2]` returns `h`, which is the second character from the right.
- `e[0:-4]` returns `Railroads`, which includes all characters excluding the last four.

In the next step, you will learn about Python's built-in functions and how those functions are used with variables. Before moving on to functions, you will comment out the existing lines of code.

❑ In the BasicsOfPython.py script window:

- Highlight all executable lines of code in your script.
- Right-click and choose Source code > Comment out region (as shown below).



```
# Name:
# Date:
# Purpose: This script i
a = 5
print a

b = 5 + 6
print b

c = "Hello"
print c

d = ["Jack", "Diane", "Lisa"]
print d[0]
```

Two gray pound signs (`##`) are placed in front of each line of code.

❑ Verify that the last `print` statement is commented out. If you did not press Enter after the last line of code, the line will not be commented out.

All of the code in the BasicsOfPython.py script is now unexecutable

When writing scripts, there will be times when you want to combine two strings together. You can do this using the plus sign (+).

- ❑ In the BasicsOfPython.py window, after the `print e[0:9]` statement, type the following code:

```
f = "Streets"
g = ".shp"
print f + g
```

- ❑ Run the script.

The two string values are concatenated together and `Streets.shp` is printed to the Interactive Window.

At times, you might need to concatenate a string value with a non-string value.

- ❑ In the BasicsOfPython.py window, after the print `f + g` statement, type the following code:

```
h = 1
print f + h + g
```

- ❑ Run the script—you will get an error.

An error is returned because you are trying to concatenate string values with non-string values. You can use the `str` built-in function to change a non-string value to a string value. You will learn more about Python built-in functions in the next exercise step.

- ❑ Modify the last `print` statement to the following:

```
h = 1
print f + str(h) + g
```

- ❑ Run the script.

The three string values are concatenated together and `Streets1.shp` is printed to the Interactive Window.

- ❑ Comment out all executable lines of code.

Stop here. Proceed to the next step when instructed.

Step 3: Explore built-in functions

Python has a number of built-in functions that allow you to perform various operations. The `round` function is just one of many built-in functions.

- ❑ In the BasicsOfPython.py window, at the end of the script, type the following code:

```
i = round(3.4)
print i
```

- ❑ Run the script.

The value of `3.4` is rounded to `3.0`.

You can access the list of built-in functions using the `dir(__builtins__)` statement.

- ❑ In the BasicsOfPython.py window, after the `print i` statement, type the following code—note that there are two underscores before the word `builtins` and two underscores after it:

```
print dir(__builtins__)
```

- ❑ Run the script.

1. Do you see the round function in the returned list?

____

There are several dozen built-in functions that are available with Python. You can use any of these functions in a script. Python, however, is not limited to this list of built-in functions. In the next step, you will learn how to access additional functions.

Stop here. Proceed to the next step when instructed.

Step 4: Work with modules

In addition to the set of built-in functions available with Python, there are hundreds of additional functions stored in modules. To access a module, you must first import the module. In this step, you will type all your code in the Interactive Window instead of in the BasicsOfPython.py script window.

Note: Most code that you type in a script can be typed into the Interactive Window as well. The main difference is the code that you type in the Interactive Window is not saved to a script. Therefore, once you exit PythonWin, you will no longer have access to that code.

The first module that you will import is the `math` module.

- ❑ In the Interactive Window, verify that you have the >>> prompt. If not, press Enter.
- ❑ In the Interactive Window, type the following code:

```
>>> import math
```

Do not type the three arrows (>>>). When you see the three arrows in documentation, it indicates that the code should be written in the Interactive Window.

- ❑ Press Enter.

This statement provides access to all the functions in the `math` module, but does not list them. You will use the `dir` statement in the `math` module to list the functions.

- ❑ In the Interactive Window, type the following code and press Enter.

```
>>> dir(math)
```

A series of functions are available from the `math` module.

2. Is the `sqrt` function part of the `math` module.

____

3. Is the `add` function part of the `math` module.

____

You can learn the definition and syntax of a function by using the `__doc__` statement. Note that there are two underscores before and two underscores after the word `doc`.

❑ In the Interactive Window, type the following code and press Enter.

```
>>> print math.sqrt.__doc__
```

Note: The `print` statement is not required in the Interactive Window, but the output is formatted better if you use the `print` statement.

The Interactive Window shows the syntax and the definition of the `sqrt` function. You can use this information to properly code the `sqrt` function.

- Syntax: `sqrt(x).`
- Definition: `Return the square root of x.`

❑ In the Interactive Window, type the following code and press Enter.

```
>>> math.sqrt(64)
```

The value of `8.0` is returned.

Notice that you need to prefix non-built-in functions with the name of the module. This makes your code more readable because you will always know where a function is coming from.

The `random` module is another module you can import.

❑ Using the skills that you just learned, import the `random` module and obtain a list of all the functions in the `random` module.

4. Is the `choice` function available from the `random` module.

____

The `help` statement can also display the syntax and definition for a function.

❑ In the Interactive Window, type the following code and press Enter.

```
>>> help(random.choice)
```

5. What is the syntax for the `choice` function?

____

6. What is the definition of the `choice` function?

____

- ❑ In the Interactive Window, type the following code and press Enter.

```
>>> random.choice(["a", "b", "d", "z"])
```

The choice function returns a random item from the list.

- ❑ In the Interactive Window, press Ctrl+Up Arrow to recall the command, then press Enter.
- ❑ Repeat a few times to see that the value returned from the list is truly random.

The `math` and `random` modules are just two of the modules that are available to import. There are dozens of other modules available.

- ❑ To view a list of the modules, in the Interactive Window, type **`help ('modules')`** and press Enter.

A listing of all available modules prints to the Interactive Window (after a brief delay).

Stop here. Proceed to the next step when instructed.

Step 5: Make decisions

One of the most powerful aspects of scripting is the ability to use decision-making statements. A Python decision-making statement that you will frequently use in your code is `if-elif-else`.

- ❑ In the BasicsOfPython.py window, comment out all executable lines of code.
- ❑ At the end of the script, type the following code:

Indentation is a language construct in Python, so use the same indentation.

```
x = 5
if x < 5:
     print "The number is less than 5"
elif x > 5:
     print "The number is greater than 5"
else:
     print "The number is equal to 5"
```

- ❑ Press Enter twice to dedent your cursor.

- ❑ Run the script.

Currently, the value of `x` is equal to `5`; therefore, the `else` statement gets executed and the Interactive Window reports, `The number is equal to 5`.

- ❑ If you have time, change the value of `x` to a number greater than `5`, then a number less than `5`, and rerun the script.

Best practice: Python's decision-making statements

- `if`, `elif`, and `else` must be lowercase.
- Use a colon (:) at the end of each condition.
- What executes for each condition is based on indentation; there is no `endif` statement. In other words, as soon as you dedent your code, Python interprets that as the end of the condition.

You will explore working with loops in the next step.

Stop here. Proceed to the next step when instructed.

Step 6: Work with loops

Another powerful aspect of scripting is the ability to run a series of statements over and over again. This is called a loop. Python has three types of loops: while, for, and range. In this step, you will work with all types of loops.

- ❑ In the BasicsOfPython.py window, at the end of the script, type the following code:

Indentation is a language construct in Python, so use the same indentation.

```
y = 1
while y < 10:
     print y
     y = y + 1
```

- ❑ Press Enter twice to dedent.

These lines of code represent a while loop. In a while loop, the loop continues to execute while the condition is true. Each line of code is described below:

- Line 1: `y` is initially assigned a value of `1`.
- Line 2: As long as `y` is less than `10`, enter the loop.
- Line 3: Print the value of `y` to the Interactive Window.
- Line 4: Add the value of `1` to `y` and go back to the top of the loop (Line 2).

The while loop will continue to execute until `y` is greater than or equal to `10`. Once `y` reaches a value of `10`, the loop will terminate.

7. How many numbers do you expect to be printed to the Interactive Window?

____

- ❑ Run the script.

The second type of loop is the list loop. A list loop will loop over each value in a list until the list is empty.

- ❑ In the BasicsOfPython.py window, at the end of the script, type the following code:

```
fcList = ["City.shp", "Roads.shp", "Railroads.shp"]
for eachFC in fcList:
     print eachFC
```

- ❑ Press Enter twice to dedent.

These three lines of code represent a list loop. In this example, the list loop will run three times, once for each value in the list.

- ❑ Run the script.

The name of each shapefile is printed to the Interactive Window.

The third type of loop is the counted loop. A counted loop will loop over a range of values.

- ❑ In the BasicsOfPython.py window, at the end of the script, type the following code:

```
for num in range(3,7):
     print num
```

- ❑ Press Enter twice to dedent.

These two lines of code represent a counted loop. In this example, the loop will run four times, once for each value in the range. The last number in the range is never included.

- ❑ Run the script.

Values `3`, `4`, `5`, `6` are printed to the Interactive Window.

Best practice: Python's looping statements

- `while`, `for`, `in`, and `range` must be lowercase.
- Use a colon (:) at the end of each `while` or `for` statement.
- What executes for each loop is based on indentation. In other words, as soon as you dedent your code, Python interprets that as the end of the loop.

- ❑ Close the BasicsOfPython.py window.

Best practice: General tips

- Comment your code.
- Variable names, statements, and functions are case sensitive.
- Geoprocessing function and class names are case sensitive.
- Pathnames are not case sensitive.

Challenge: Work with functions

Estimated time: 10 minutes

Write all the code for this step in the Interactive Window.

- ❑ Print the length of the following string to the Interactive Window:
  `"SanDiego.gdb"`

- ❑ Print the basename of the following path to the Interactive Window:
  `"C:\\Student\\PYTH\\Database\\Redlands.gdb\\Parcels"`

- ❑ Print the following string, minus the last four characters, to the Interactive Window:
  `"Parcels.shp"`

Python resources

Python offers a lot of functionality of which only a small part is covered in this course. If you want to learn more about Python syntax and functionality, you can purchase one of the many books available on the market or go to Python's Web site to find helpful resources.

Books

- *Learning Python (4th edition)*, by Mark Lutz and David Ascher
- *Learn to Program Using Python*, by Alan Gauld
- *Python in a Nutshell (2nd edition)*, by Alex Martelli
- *Python Cookbook (2nd Edition)*, by Alex Martelli, Anne Martelli Ravenscraft, and David Ascher

Web sites

- **http://www.python.org**
  Tutorials, documentation, and forums
- **http://diveintopython.org**
  Dive Into Python, by Mark Pilgrim
- **http://openbookproject.net/thinkCSpy/**
  How to Think Like A Computer Scientist: Learning with Python (2nd Edition) , by Jeffrey Elkins, Allen B. Downey, and Chris Meyers
- **http://sourceforge.net/projects/pywin32/**
  PythonWin installation
- **http://code.activestate.com/recipes/langs/python**
  Extensive collection of Python scripts (very few GIS scripts)

Lesson review

1. Is PythonWin installed by ArcGIS?

2. What is a variable in Python?

3. In PythonWin, where can you go to get more information about a module?

4. Write a line of Python code that prints the built-in functions to the Interactive Window.

5. Fix the three Python syntax errors below.

```
if x = 5:
    print "x is equal to 5"
elif x > 5
    print "x is greater than 5"
else:
    print "x is not equal to or greater than 5'
```

Python reference

Variables	A variable is simply a name that represents a value. To create the variable, assign a value to the name. An example might be `x = 3`. To evaluate the value, simply refer to the name as in `print x`. Some rules to be aware of when working with variables: ▪ Variable names are case sensitive. `Scale` is a totally different variable than `scale`. ▪ Variables are dynamically data typed in Python, so the data type is determined when the variable is referenced. ▪ Variables can hold the data types of String, Number, Lists, Tuples, Dictionaries, Files, and objects.
Strings	The String data type simply stores a string of characters, which could include both characters and numbers. The string is enclosed by either single or double quotes and Python does not care which one you use, as long as they match both sides of the value being assigned to the string variable. This provides the ability to embed one string inside another. When a string contains a pathname, you can use two backslashes or one forward slash to indicate the directories. Strings can be concatenated together with the use of the **'+'** symbol. Strings are indexed, where each character in the string is assigned an index number. The first index number starts with 0 and increments up. This gives you the ability to slice the value. As an example, you write a line of code to assign a value to a string, such as `fc = "Streets.shp"`. If you wanted to get the first three characters, you would write `fc[0:2]`, which would return **'Str'**. If you wanted all characters but the last three, you would write `fc[0:-4]`, which would return **'Streets'**. To get all characters from the third to the end, you would write `fc[2:]`, which would return **'reets.shp'**.
Numbers	The number data type can store numbers and expressions. Numbers are distinguished from strings because they are not surrounded by quotes. Numbers can represent something as simple as a single digit to something as complex as the return value of a mathematical expression.

Python Lists	In Python, the most basic form of a data structure is a sequence. Each element of the sequence is assigned a number (or index). The first data element startes with 0, the second with 1, the third with 2, and so forth. A Python List stores the sequence enclosed by square brackets, such as `mydata = [1,2,3,4,5,"a"]` . Python List elements are accessed by their index number, so `print mydata[5]` will display **'a'** to the Interactive Window in PythonWin. Python Lists can be manipulated so that the data elements can be changed, reordered, removed, the order of the elements reversed, sorted, new elements added to the end of the list, or inserted at a specified location. To determine the number of elements in the Python list, you can use the Python List method **count**, or the Python built-in function `len()`.
Dictionaries	A dictionary is an unordered collection of data elements that are stored in the form of a key:value pair combination. The key provides a name for it's paired value, so that when we want to get a particular data element, we refer to the name and the value is retrieved. A key can be either a string or a number. To populate the dictionary, assign the key:value pair values surrounded by curly brackets to the dictionary, such as `dataDict = {"x": 6765992, "y": 133454.6}`. To access the **'x'** key value, evaluate it as `print dataDict["x"]`.
Tuples	Tuples store sequences just like Python Lists, but they are not changeable. Just like Python Lists, the elements in the tuple are accessed by their index number. To store elements in the tuple, assign a comma separated sequence of values, such as `x = 1, 2, 3, 4`. Elements in the tuple can be sliced just like a Python List or string.

Python's decision-making syntax is if…elif…else.

At the end of each condition, a colon is required. If you forget the colon, you will get a syntax error. To finish a condition, dedent the code. There is no statement to end the decision-making block. As soon as you dedent your code, Python knows that the decision-making block has ended.

Indentation is a language construct in Python.

You can use any number of spaces or tabs for indentation—just make sure you are consistent within any given block of code. To ensure that the proper code gets indented, Python automatically indents the code when you press Enter. The only thing you need to remember is to dedent the code.

Why does Python use indentation?

- It is one less line of code that you need to write. Python tries very hard to minimize the lines of code required to perform any task. This makes Python scripts easy to maintain.
- More importantly, indentation makes a script more readable. Most scripting and programming languages encourage people to indent blocks of code to make the code more readable, but it is not widely enforced and people get sloppy. Python forces the programmer to indent these blocks of code, which ensures readable, well-formatted code.

Line continuation

Lines of code can become very long. To make your code more readable, continue long lines of code onto another line. This will ensure that the user does not need to scroll through the code.

There are a number of line continuation characters in Python.

- The backslash (\) character at the end of a line will continue a line onto the next line.
- Enclosing values in parentheses, brackets, or braces will also continue a line onto the next line.

Either way will work, but it is more common today for Python programmers to use parentheses, brackets, or braces than the backslash. As long as you use parentheses, brackets, braces, or a backslash, Python recognizes the line continuation and automatically indents your code when you press Enter.

Note: Python recognizes values enclosed in brackets [ ] as lists; values enclosed in parentheses ( ) or enclosed in nothing as tuples; and values enclosed in braces { } as dictionaries.

Answers to Lesson 1 questions

Exercise 1: Learn the basics of Python

1. Do you see the round function in the returned list?

   Yes

2. Is the `sqrt` function part of the `math` module.

   Yes.

3. Is the `add` function part of the `math` module.

   No.

4. Is the `choice` function available from the `random` module.

   Yes.

5. What is the syntax for the `choice` function?

   `choice(self, seq)`

6. What is the definition of the `choice` function?

   Choose a random element from a non-empty sequence.

7. How many numbers do you expect to be printed to the Interactive Window?

   Numbers 1 to 9. The number 10 does not get printed because the loop only executes if the value is less than 10.

Lesson review

1. Is PythonWin installed by ArcGIS?

   No.

2. What is a variable in Python?

   A named location in memory that can store a value.

3. In PythonWin, where can you go to get more information about a module?

   Help > Global Module Index

4. Write a line of Python code that prints the built-in functions to the Interactive Window.

   `dir(__builtins__)`

5. Fix the three Python syntax errors below.

```
if x = 5:
    print "x is equal to 5"
elif x > 5
    print "x is greater than 5"
else:
    print "x is not equal to or greater than 5'
```

- **Line 1 is missing an equal sign (=). In Python, a single equal sign means to assign a value and a double equal sign means to evaluate the value.**
- **Line 3 is missing a colon (`:`). All conditional statements must end with a colon.**
- **The last line is using a single quotation mark (`'`) at the end of the line instead of a double quotation mark (`"`). Quotation marks must be balanced: if you use a single quotation mark at the start of a text string, it must end with a single quotation mark. The same logic applies for double quotation marks.**

***Solution*:**

```
if x == 5:
    print "x is equal to 5"
elif x > 5:
    print "x is greater than 5"
else:
    print "x is not equal to or greater than 5"
```

Challenge solution: Work with functions

```
# Print the length of the following string
# to the Interactive Window: "SanDiego.gdb"

>>> len ("SanDiego.gdb")

# Print the basename of the following path to the Interactive Window:
# "C:\\Student\\PYTH\\Database\\Redlands.gdb\\Parcels"

>>> import os.path
>>> os.path.basename("C:\\Student\\PYTH\\Database\\Redlands.gdb\\Parcels")

# Print the following string, minus the last four characters,
# to the Interactive Window: "Parcels.shp"

>>> "Parcels.shp"[0:-4]
```

Exercise solution

BasicsOfPython.py

```
# Name: ESRI
# Date:
# Purpose: This script is an exercise to learn the Python scripting language.

a = 5
print a

b = 5 + 6
print b

c = "Hello"
print c

d = ["Jack", "Diane", "Lisa"]
print d[0]
print d[2]
print d[-1]

e = "Railroads.shp"
print e[3]
print e[-2]
print e[0:-4]

f = "Streets"
g = ".shp"
print f + g

h = 1
print f + str(h) + g

i = round(3.4)
print i

print dir(__builtins__)
```

```
x = 5
if x < 5:
    print "The number is less than 5"
elif x > 5:
    print "The number is greater than 5"
else:
    print "The number is equal to 5"

y = 1
while y < 10:
    print y
    y = y + 1

fcList = ["City.shp", "Roads.shp", "Railroads.shp"]
for eachFC in fcList:
    print eachFC

for num in range(3, 7):
    print num
```

2

ArcPy: What's the big deal?

Introduction

Working with geoprocessing functionality has come a long way since it was first introduced. Geoprocessing functionality could be accessed by running tools in ArcToolbox, creating a model in ModelBuilder, or running the geoprocessing tool by its name in a Command Line window.

ArcPy is a site package that provides useful and productive ways to perform geographic data analysis, data conversion, data management, and map automation with Python.

In this lesson, you will explore the ArcPy site package in the ArcGIS Desktop Help and access ArcPy in both ArcGIS Desktop and in PythonWin.

Learning objectives

After completing this lesson, you will be able to:

- Describe the ArcPy site package
- Write scripts using ArcPy in PythonWin
- Access ArcPy in the Python window
- Work with ArcPy modules

Key terms

- **ArcPy**
- **ArcPy modules**
- **ArcPy classes**
- **ArcPy functions**

Note: For definitions, go to the ArcGIS Desktop Help, and navigate to:

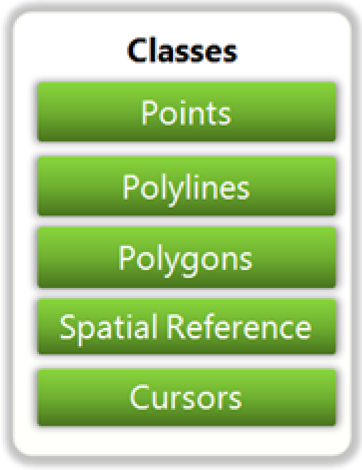
- Professional Library >
- Geoprocessing >
- The ArcPy site package >
- Essential ArcPy vocabulary

What does geoprocessing mean to you?

Notes

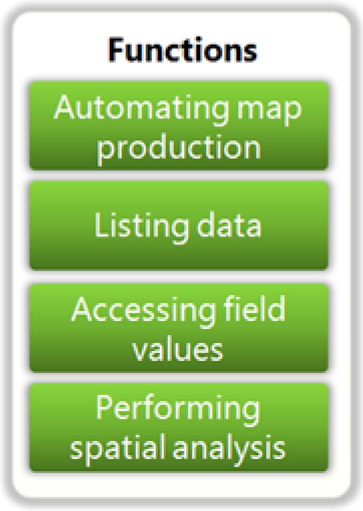
ArcPy functions and classes

ArcPy contains many classes and functions for the Python scripter.



A *class* is a blueprint for creating an object in Python.

When you work with an ArcPy class, the object that you create will exist only in memory and cease to exist once the script completes or when you close the Python IDE. The advantage is that intermediate storage on disk is not needed in order to use the object.



A *function* is a bit of defined functionality that can be accessed in a script. In ArcPy, all geoprocessing tools are functions. In the Help, functions are grouped by the tasks they perform.

An ArcPy function typically takes an argument as a parameter, performs the task, and returns a result. You can use the result returned from an ArcPy function further down in your script. For example, you run the Clip tool from the Analysis toolbox, then pass in the result as one of the inputs to the Union tool in the Analysis toolbox.

ArcPy classes can create geometry objects such as Points, Polylines, and Polygons, as well as SpatialReference or Extent objects. These objects can be used to create and update features, or to represent feature classes, tables, fields, or spatial references. Other types of objects that can be created from ArcPy classes include Field, Raster, Row, and FeatureSet objects, just to name a few.

ArcPy functions support certain geoprocessing workflows. For example, you can:

- List data for automating processing of datasets.
- Describe data properties for decision-making in a script.
- Step through features in a feature class or rows in a table using a cursor data access object for reporting, updating field values, or creating new data.
- Make sure the Catalog window in ArcMap reflects the latest changes to data.
- Update layers in your map document.
- Set and/or receive parameter values in your scripts.
- Provide reports about the ArcGIS Desktop installation on a computer.
- Add messages to the progress dialog and update the progress dialog status bar.
- Add your custom toolbars to the Python session.

The ArcPy modules

A *module* is a Python file that typically contains task-related functions and classes. ArcPy is supported by a series of modules.

Mapping module	**`arcpy.mapping`**	Provides automation for map document (.mxd) management
Spatial Analyst module	**`arcpy.sa`**	Provides access to the Spatial Analyst functions and operators
Geostatistical Analyst module	**`arcpy.ga`**	Provides access to the Geostatistical Analyst functions

Explore the help topics about these ArcPy modules.
In the ArcGIS Desktop Help (Start > All Programs > ArcGIS), expand:

- Professional Library >
- Geoprocessing >
- The ArcPy site package >
  - Mapping module
  - Spatial Analyst module
  - Geostatistical Analyst module

1. If your geodatabase was moved to a new folder or drive location and you had several map documents that pointed to the older location, which ArcPy module should you import in your script so that you can repair the map documents?

   ____

2. How can you import all of the ArcPy sub-modules into your script?

   ____

You can also import each sub-module separately.

```
from arcpy import mapping  # Imports the mapping sub-module
mapping.AddLayer(...)

import arcpy.ga as GA
GA.CrossValidation(...)
```

Additional notes

arcpy.mapping	Contains functions and classes for working with map documents (MXD) to fix and repair layers and workspaces, report contents of the MXD, print out map series for map book creation, and export the maps to many graphic formats including PDF, TIFF, JPEG, AI, and PNG.
arcpy.sa	Provides a shortcut to the Spatial Analyst tools, functions, operators and classes for working with rasters and surfaces. An advantage to using this module is that the geoprocessing tool, function, or class can be more easily referenced in the script by a shortened name. For example: `from arcpy.sa import *` `FocalStatistics(<in_raster>, <neightborhood_raster>, ...)`
arcpy.ga	Provides access to the Geostatistical Analyst classes, which are primarily used to define parameters for Geostatistical Analyst tools that may have a varying number of arguments depending on the parameter type selected (e.g., the search neighborhood). By using classes for parameters, you can access and programmatically change any of the individual entries in the parameter within a model script.

Exercise 2: Working with ArcPy

Estimated time: 20 minutes

In this exercise, you will access ArcPy in both the ArcMap Python window and in a script that you will write in PythonWin.

The Corvallis school district has approached you to create a new crime analysis feature class. The requirements are that the schools are to be buffered by 500 meters, but the buffers must not extend outside the city boundary line by more than 1500 meters.

You decide that the best approach is to create the school buffers using the ArcMap Python window first, then create a new script in PythonWin and test run it. When your new script runs successfully in PythonWin, you will run the same script in the ArcMap Python window and view the results. Finally, you will access the Clip_analysis tool from the Python window to clip the school buffers against the buffered boundary line for the final output.

In this exercise, you will:

- Run the Buffer tool in the Python window
- Create a script in PythonWin
- Run the script in the Python window

Step 1: Access ArcPy in ArcMap

In this step, you will open an existing MXD in ArcMap and open the Python window. You will then access the Buffer_analysis tool and view the tool syntax in the Python window help panel. After running the tool, you will examine the results in the Table of Contents.

- ❑ Open ArcMap. (From the Start menu, choose All Programs > ArcGIS > ArcMap.)

- ❑ In the ArcMap - Getting Started dialog box, navigate to Existing Maps > Browse for more.

- ❑ In the Open ArcMap Document dialog box, browse to C:\Student\PYTH\Exercise02, select Corvallis.mxd, and click Open.

- ❑ Open the ArcGIS Desktop Help (Click Start > All Programs > ArcGIS > ArcGIS Desktop Help > ArcGIS Desktop 10 Help).

- ❑ In the ArcGIS 10 Help window, on the Contents tab, expand the following:
  - Professional Library >
  - Geoprocessing >
  - Executing tools >
  - Executing tools using the Python window >
  - Using the Python window
- ❑ Select *Executing tools in the Python window* and review the contents of this topic to answer the following questions.

1. Where can you find the tool name for a geoprocessing tool that you want to run in the Python window?

   ____

   ____

2. List three ways to skip a tool's optional parameters.

   ____

3. In the Python window, how do you get help for a geoprocessing tool?

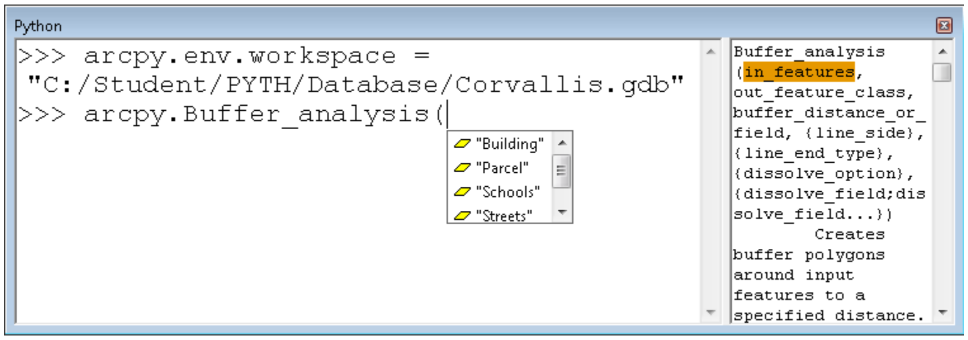
   ____

   ____

- ❑ Close the ArcGIS Help window.
- ❑ In ArcMap, from the Geoprocessing menu, choose Python.
- ❑ In the Python window that opens, type in the following code:

```
arcpy.env.workspace = "C:/Student/PYTH/Database/Corvallis.gdb"
arcpy.Buffer_analysis(
```

> **Note:** You do not need to import the ArcPy site package when using the Python window.



Python

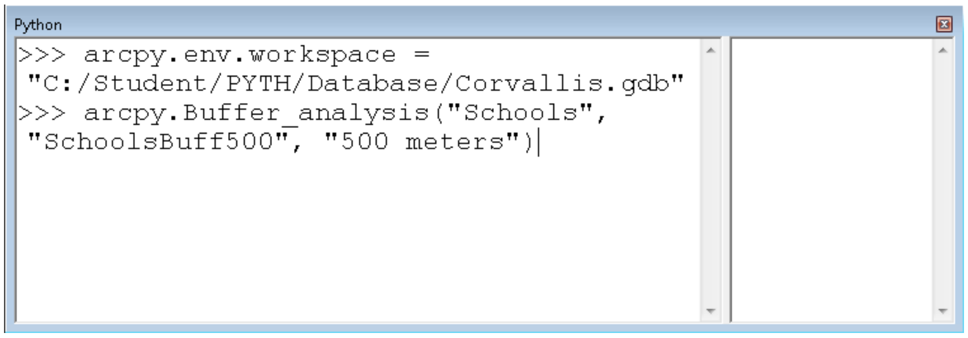
```
>>> arcpy.env.workspace =
 "C:/Student/PYTH/Database/Corvallis.gdb"
>>> arcpy.Buffer_analysis(
```

"Building"
"Parcel"
"Schools"
"Streets"

```
Buffer_analysis
(in_features,
out_feature_class,
buffer_distance_or_
field, {line_side},
{line_end_type},
{dissolve_option},
{dissolve_field;dis
solve_field...})
          Creates
buffer polygons
around input
features to a
specified distance.
```

Notice that the syntax for the Buffer tool displays in the Help and Syntax panel of the Python window.

❑ Using the displayed syntax, enter the required parameters to buffer 500 meters from the Schools and create the **SchoolsBuff500** feature class.



Python

```
>>> arcpy.env.workspace =
 "C:/Student/PYTH/Database/Corvallis.gdb"
>>> arcpy.Buffer_analysis("Schools",
 "SchoolsBuff500", "500 meters")
```

❑ Press Enter to run the tool.

Notice that by default the newly created output dataset is added to ArcMap as a layer.

❑ Leave ArcMap open. You will return to the Python window in a later step.

Step 2: Access ArcPy in PythonWin

The ArcPy site package can be accessed outside of ArcGIS Desktop in any Python IDE. In this step, you will write a short Python script in PythonWin to import the ArcPy site package, set the workspace environment setting, and run the `Buffer_analysis` tool on the city boundary feature class.

❑ If necessary, start PythonWin from the shortcut on your desktop or from the taskbar.

> **Note:** PythonWin can also be started from C:\Python26\ArcGIS10.0\Lib\site-packages\pythonwin\Pythonwin.exe

- ❑ In PythonWin, create a new Python Script and save the file as **Buffer_boundary.py** in the C:\Student\PYTH\Exercise02 folder.

- ❑ In the Buffer_boundary.py window:
  - Import arcpy.
  - Set the workspace environment to the Corvallis geodatabase.
  - Use the following parameters with the Buffer_analysis tool:
    - *in_features*: **`"BoundaryLine"`**
    - *output_feature_class*: **`"Boundary1500"`**
    - *distance*: **`"1500 meters"`**
  - Print a message to the Interactive Window when the script is complete.

- ❑ Save your script and click the Run button.

- ❑ In the Run Script dialog box, click the OK button to run your script.

If you encounter any errors, ask your instructor for help or refer to the ArcGIS Desktop Help.

> **Note:** When applicable, the workbook includes exercise solutions at the end of the lesson.

- ❑ Close PythonWin.

It is always a good idea to verify that your script has not only completed successfully but that it also ran correctly.

Next, you will verify that the new Boundary polygon feature class has been created.

- ❑ Open ArcCatalog and navigate to C:\Student\PYTH\Database\Corvallis.gdb and expand or select it.

- ❑ The new Boundary1500 feature class should be listed.

- ❑ Close ArcCatalog and return to ArcMap.

- ❑ In ArcMap:
  - Click the Add Data button.
  - Browse to C:\Student\PYTH\Database\Corvallis.gdb.
  - Add theBoundaryLine and Boundary1500 feature classes to the map.
  - Verify that the Buffer tool created Boundary1500 correctly.

Step 3: Run a Python script in the Python window

Your Python scripts can also be run directly in the Python window. In this step, you will load and run your script directly in the Python window in ArcMap.

One of the geoprocessing options that you can set is the option to overwrite the output of geoprocessing operations. You will set this option now.

- ❑ From the Geoprocessing menu, choose Geoprocessing Options.
- ❑ In the Geoprocessing Options dialog box, check the option to Overwrite the outputs of geoprocessing operations, then click OK.

Now you will load your script and run it again.

- ❑ Open the Python window, if necessary.
- ❑ In the Python window, right-click and select Load.
- ❑ In the Open dialog box, navigate to C:\Student\PYTH\Exercise02, select Buffer_boundary.py, and click Open.
- ❑ In the Python window, press the Enter key to run the script.

You should not encounter any errors.

4. If you encountered errors or if the features in the new Boundary1500 layer are not correct, what steps should you perform?

____

____

Challenge: Run the Clip tool in the Python window

Estimated time: 5 minutes

Finally, you will clip the school buffers to the Boundary1500 polygon.

- ❑ In the Python window, use the Clip analysis tool to clip the SchoolsBuff500 layer by the Boundary1500 layer to produce the new ClippedSchoolsBuffer layer.
- ❑ Verify the results in ArcMap.
- ❑ Close ArcMap and do not save any changes to Corvallis.mxd.

Lesson review

1. How can you access geoprocessing functionality in Python scripts?

2. What is the ArcPy site package?

3. For what might you use the arcpy.mapping module?

4. List some advantages to using ArcPy.

Creating the geoprocessor

A little bit of history

Since ArcGIS version 9.0, Python has been available to script geoprocessing tasks. In the early days of geoprocessing and scripting, the scripting environment was limited to Windows. At version 9.2, Python and scripting of geoprocessing tools was opened to Unix/Linux machines and ArcGIS Server.

ArcGIS 10

ArcGIS 10 introduces the ArcPy site package for Python, which integrates Python into the Desktop world. You can choose to run your geoprocessing scripts in any of these ways:

- In a Python Integrated Development Environment application
- Schedule the script to run at a certain time
- From ArcToolbox as a tool
- From a menu in ArcGIS Desktop
- As a background process in ArcGIS Desktop
- In the Python window

ArcGIS 10 supports backward compatibility with scripts that use the `arcgisscripting` module.
You can create the geoprocessor object using the ArcGIS 9.2 or 9.3 `create` method, but not all ArcPy functionality will be available in your script.

Creating the geoprocessor object

ArcGIS version	Python code
10	`import arcpy`

9.3	`import arcgisscripting`
	`gp = arcgisscripting.create(9.3)`

9.2	`import arcgisscripting`
	`gp = arcgisscripting.create()`

9.0, 9.1	`import win32com.client`
	`gp = win32com.client.Dispatch("esriGeoprocessing.GpDispatch.1")`

Answers to Lesson 2 questions

The ArcPy modules

1. If your geodatabase was moved to a new folder or drive location and you had several map documents that pointed to the older location, which ArcPy module should you import in your script so that you can repair the map documents?

   The arcpy.mapping module contains the functions to fix and repair layers in your map documents.

2. How can you import all of the ArcPy sub-modules into your script?

   ```
   import arcpy
   ```

Exercise 2: Working with ArcPy

1. Where can you find the tool name for a geoprocessing tool that you want to run in the Python window?

   In the tool properties, which you can access in the Catalog or ArcToolbox windows. Unlike the tool label (which displays in ArcToolbox and at the top of the tool's dialog box), the tool name does not contain spaces.

2. List three ways to skip a tool's optional parameters.

   Use an empty set of quotation marks, a pound sign within quotation marks, or specify the parameter name.

3. In the Python window, how do you get help for a geoprocessing tool?

   Type the geoprocessing tool name at the Python window prompt followed by an open parenthesis. The tool usage and help documentation will display in the Python window Help and Syntax panel.

4. If you encountered errors or if the features in the new Boundary1500 layer are not correct, what steps should you perform?

   Close ArcMap to remove any locks on the data, open the script in PythonWin and check for syntax or logic errors, and verify that the parameters used for the Buffer tool are correct.

Lesson review

1. How can you access geoprocessing functionality in Python scripts?

   Import the ArcPy site package.

2. What is the ArcPy site package?

   The ArcPy site package provides Python access for all geoprocessing tools, including extensions, as well as a wide variety of useful functions and classes for working with and interrogating GIS data.

3. For what might you use the arcpy.mapping module?

   The arcpy.mapping module can be used to open and manipulate ArcMap map documents (.mxd) and layer files (.lyr). Scripts that use arcpy.mapping can open map documents (.mxd) and layers, query and alter the contents, and then print, export, or save the modified document.

4. List some advantages to using ArcPy.

   ArcPy provides access to geoprocessing tools as well as additional functions, classes, and modules that allow you to create simple or complex workflows quickly and easily.

   Scripts written using ArcPy benefit from being able to access and work with any Python module.

Challenge solution: Run the Clip tool in the Python window

The arcpy.env.workspace setting is already set to the Corvallis.gdb geodatabase—the script that you loaded earlier set this, so you don't need to do that again.

- ❑ Only one line of code is needed at the Python window prompt:

```
arcpy.Clip_analysis("SchoolsBuff500", "Boundary1500", "ClippedSchoolsBuffer")
```

Exercise solution

Buffer_boundary.py

```
# Name: <your name>
# Date: <current date>
# Purpose: This script will buffer the Boundary
# polyline feature class by 1500 meters.

# Import the ArcPy site package
import arcpy

# Set the workspace environment
arcpy.env.workspace = "C:/Student/PYTH/Database/Corvallis.gdb"

# Run the Buffer tool in the Analysis toolbox
arcpy.Buffer_analysis("BoundaryLine", "Boundary1500", "1500 meters")

# Print a message to the Interactive Window
print "Script completed"
```

3 Debugging your scripts

Introduction

In this lesson, you will examine several steps in debugging code using PythonWin. Knowing how PythonWin finds syntax errors in a script can help quickly locate the problem.

Learning objectives

After completing this lesson, you will be able to:

- Debug scripts in PythonWin
- Visually find errors

Script debugging workflow

When you need to debug your script, the best environment is one that allows you to check for syntax errors, comment your code, and print values to a window, and that provides functionality to step through the script line by line and examine variables.

PythonWin and IDLE provide these capabilities. You will use PythonWin throughout this course to write and debug your scripts.

A typical script debugging workflow

1. Write the script in PythonWin and check for syntax errors.
2. Debug the script in PythonWin using the Debugging toolbar.
3. Run the script with arguments.
4. Test the script in PythonWin and check for expected results.
5. Comment blocks of code to help narrow down the error.
6. Use print statements to verify correct values for variables.
7. Run the script in the Python window or as a *script tool** in ArcGIS Desktop or ArcToolbox.

**You will learn about script tools in a later lesson.*

Notes

Activity: Finding visual errors in scripts

The instructor will show three different scripts that contain errors. A copy of each script is located here in your workbook, so that you can record the location of the errors and the fixes.

Note: Solutions are provided at the end of the lesson.

Find and correct the errors in the following scripts.

1. Find four errors.

```
# This script buffers the Railroads feature class from the
# SanDiego.gdb geodatabase. The buffer distance is 500 meters

import arcpy

arcpy.env.Workspace = "C:\Student\\PYTH\\Database\\SanDiego.mdb"

inFC = "Railroads"
distance = 500

arcpy.Buffer_analysis(inFC, newBuffFC, distance)
```

2. Find five errors.

```
# This script prints each item in the Python list to the
# Interactive Window.

listFC = ["Airports", "BusStations", "Schools", "Parcels"]

For fc in listFc:
    print fc

Print len(lstFc)
```

3. Find three errors.

```
# This script compares the square root of 26 with 6 and
# reports the comparison to the Interactive Window.

Import math
z = 25
y = 6
x = math.sqrt(z)

if x = y:
    print "x and y are the same"
elif x > y:
    print "x is greater than y"
else
    print "x is less than y"
```

Exercise 3: Handling syntax errors

Estimated time: 45 minutes

When writing scripts, it is almost impossible to have a script that never produces an error. This is especially true if it interacts with an end user.

In this exercise, you will debug a script that:

- Prints the arcpy.env.workspace directory name
- Creates a list of rasters in the directory
- Loops through the list, printing each raster name and format type

Step 1: Debug your script

In this step, you will debug the ListAndPrintRasters.py script, which contains several Python syntax errors.

❑ In PythonWin, open the C:\Student\PYTH\Exercise03\ListAndPrintRasters.py script.

❑ Save the script as **MyListAndPrintRasters.py** to C:\Student\PYTH\Exercise03.

This script has four syntax errors.

❑ On the Standard toolbar, click the Check button.

The status bar reports that there is invalid syntax in the script, and the cursor is placed on the following line of code:

```
lstRAS = arcpy.ListRasters(*)
```

1. What is invalid about this line of code?
   (*Hint:* The cursor is placed exactly where the error occurs.)

   ____

❑ Fix the error.

❑ Click the Check button again.

There is another syntax error in the script. The cursor is placed within the following line of code:

```
if dscRAS.format = "GRID":
```

2. What is invalid about this line of code?

____

- ❑ Fix the error.
- ❑ Click the Check button.

There is another syntax error in the script. The cursor is placed within the following line of code:

```
Print "The " + ras + " raster is stored in the ESRI GRID format"
```

3. What is invalid about this line of code?

____

- ❑ Fix the error.
- ❑ Click the Check button.

There is another syntax error in the code. The cursor is placed at the end of the following line of code:

```
else
```

4. What is invalid about this line of code?

____

- ❑ Fix the error.
- ❑ Click the Check button.

Congratulations! The PythonWin status bar reports that the script was checked successfully. You are now ready to run your script.

Step 2: Run the script

❑ Run your script.

A traceback error is printed to the Interactive Window. The NameError indicates that the name 'os' is not defined. This error occurs when a variable name is used before a value is assigned, or when a module is not imported before being referenced.

5. Is the `os` module referenced in your code?

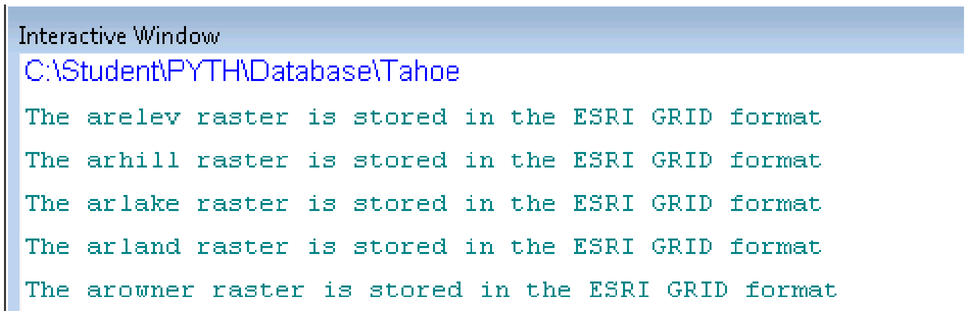
____

❑ Below the `import arcpy` statement, add a line of code that imports the **os** module.

The PythonWin Interactive Window still displays the error text. You can clean up this window by right-clicking within it and choosing Select All, then right-clicking again and choosing Cut.

❑ Click the Check button. Fix any errors that may be found.

❑ Once all errors are fixed, run the script.



```
C:\Student\PYTH\Database\Tahoe
The arelev raster is stored in the ESRI GRID format
The arhill raster is stored in the ESRI GRID format
The arlake raster is stored in the ESRI GRID format
The arland raster is stored in the ESRI GRID format
The arowner raster is stored in the ESRI GRID format
```

Your debugged script should now complete without error. Each raster name and format type prints to the Interactive Window.

❑ Close the MyListAndPrintRasters.py script.

Lesson review

1. Where do you debug your code?

2. List some common code errors.

3. List the main reason why you might receive a NameError: name 'os' is not defined in the Interactive Window.

Answers to Lesson 3 questions

Activity: Finding visual errors in scripts

1. Find four errors.

```
# This script buffers the Railroads feature class from the
# SanDiego.gdb geodatabase. The buffer distance is 500 meters

import arcpy

arcpy.env.workspace = "C:\\Student\\PYTH\\Database\\SanDiego.gdb"

inFC = "Railroads"
distance = 500

arcpy.Buffer_analysis (inFC, "newBuffFC", distance)
```

2. Find five errors.

```
# This script prints each item in the Python list to the
# Interactive Window.

listFC = ["Airports", "BusStations", "Schools", "Parcels"]

for fc in listFC:
    print fc

Print len(listFC)
```

3. Find three errors.

```
# This script compares the square root of 26 with 6 and
# reports the comparison to the Interactive Window.

Import math
z = 25
y = 6
x = math.sqrt(z)

if x == y:
    print "x and y are the same"
elif x > y:
    print "x is greater than y"
else:
    print "x is less than y"
```

Exercise 3: Handling syntax errors

1. What is invalid about this line of code?
   (*Hint:* The cursor is placed exactly where the error occurs.)

   The * is not surrounded by quotation marks.

2. What is invalid about this line of code?

   Since the code is evaluating the value of dscRAS.format, a double-equal sign must be used.

3. What is invalid about this line of code?

   The Python keyword `print` is capitalized.

4. What is invalid about this line of code?

   The Python keyword `else` is missing a colon after it.

5. Is the `os` module referenced in your code?

   Yes, the line of code that prints the directory path uses `os.path.dirname`

Lesson review

1. Where do you debug your code?

   Test and debug your scripts in PythonWin

2. List some common code errors.

   Spelling, missing colons, improper indentation, missing backslash in path name, improper variable name case (Scale vs scale)

3. List the main reason why you might receive a NameError: name 'os' is not defined in the Interactive Window.

   The os module has not been imported in the script.

Exercise solution

MyListAndPrintRasters.py

```
# Author: ESRI
# Date: <Today>
# Purpose: This script prints the names of the rasters located in the
#          Tahoe/All folder and lists the raster format type

# Import the ArcPy site package and set the workspace location
import arcpy
import os
arcpy.env.workspace = r"C:\Student\PYTH\Database\Tahoe\All"

# Print the directory path of the workspace location to the
# Interactive Window
print os.path.dirname(arcpy.env.workspace)

# arcpy.ListRasters() function will return a Python List of raster names
# You will learn more about the ArcPy List functions in Lesson05
lstRAS = arcpy.ListRasters("*")

# Iterate through the Python List, printing the raster name and
# if the raster is stored inthe GRID format.
# You will learn more about the ArcPy Describe function in Lesson04
for ras in lstRas:
    dscRAS = arcpy.Describe(ras)
    if dscRAS.format == "GRID":
       print "The " + ras + " raster is stored in the ESRI GRID format"
    else:
        print "The " + ras + " raster is not stored in the ESRI GRID format"
```

4 Using Describe objects

Introduction

The `Describe` function on the ArcPy site package is a function that returns a `Describe` object that provides descriptive information about the type of data being described. The types of data that can be described include geodatabases, geodatabase tables, feature classes, rasters, shapefiles, folders, coverages, layer files, etc.

Each property of the object that the `Describe` function returns can be used to control the flow of a script. For example, a script might ask a user for a feature class to buffer. Depending on the `ShapeType` of the feature class, the parameter values for the `Buffer` tool will differ.

Learning objectives

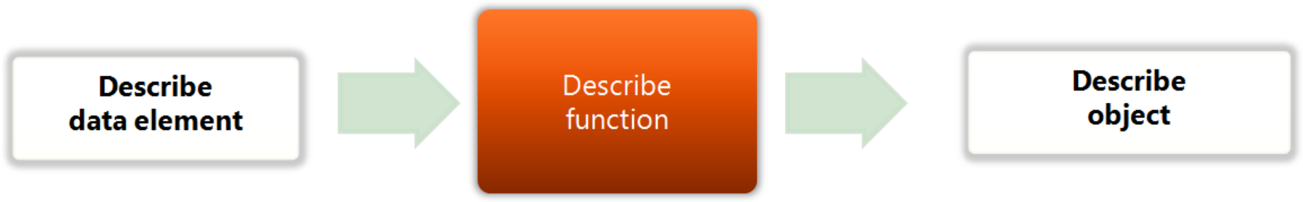
After completing this lesson, you will be able to:

- Access data properties using the Describe function
- Perform geoprocessing on data using Describe objects

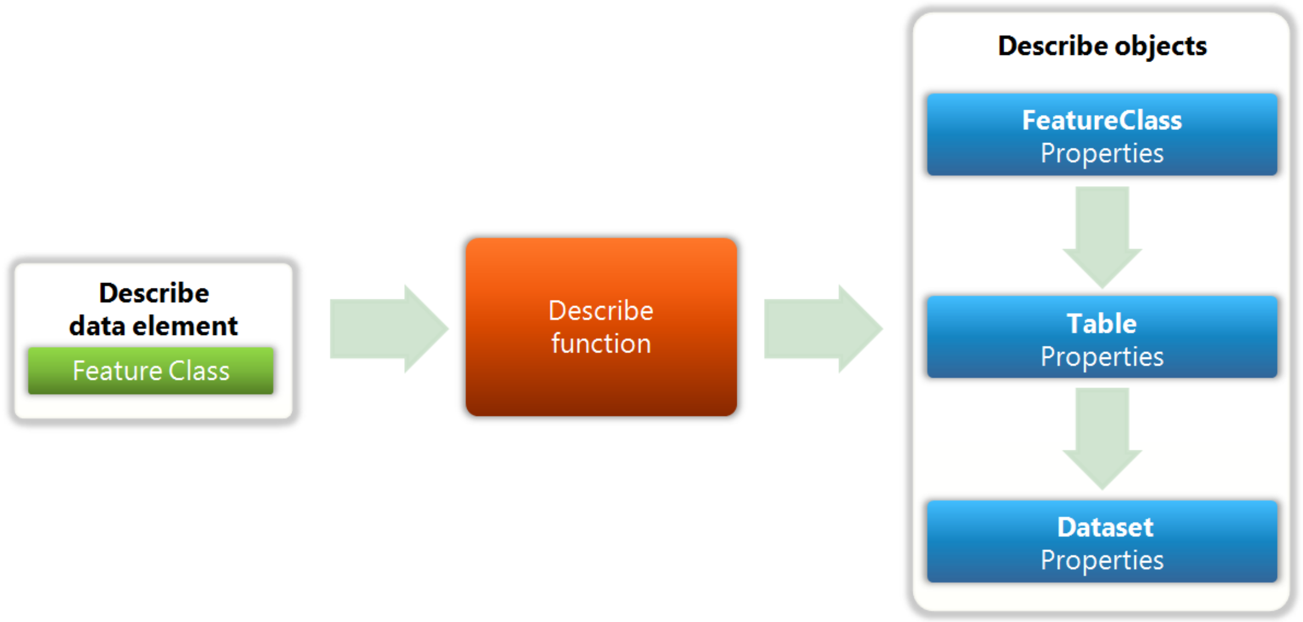
The Describe function

The ArcPy Describe function is a helper type function on the Geoprocessor. It returns a Describe object that contains descriptive properties about what is being described. This could include a folder, geodatabase, feature class, table, raster, coverage, or layer file.

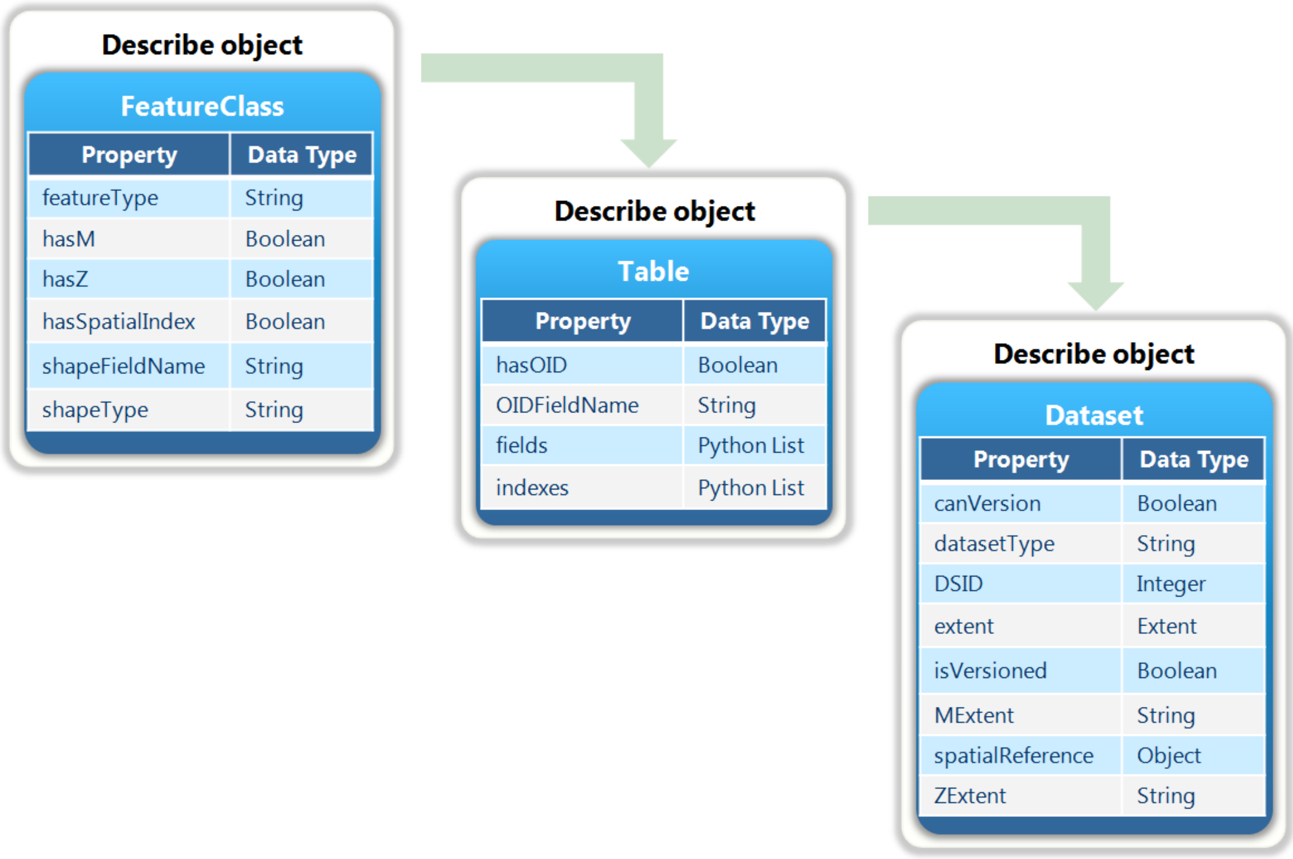
We can use these properties to control the flow of a script or to make some sort of decision. Let's say we have a custom Buffer tool. The user inputs the feature class to buffer and the script will set the appropriate parameters for the Buffer tool, or depending on the input data the user supplied, possibly might not even call the Buffer tool.



Describing a feature class



Feature class properties



Describe object

FeatureClass

Property	Data Type
featureType	String
hasM	Boolean
hasZ	Boolean
hasSpatialIndex	Boolean
shapeFieldName	String
shapeType	String

Describe object

Table

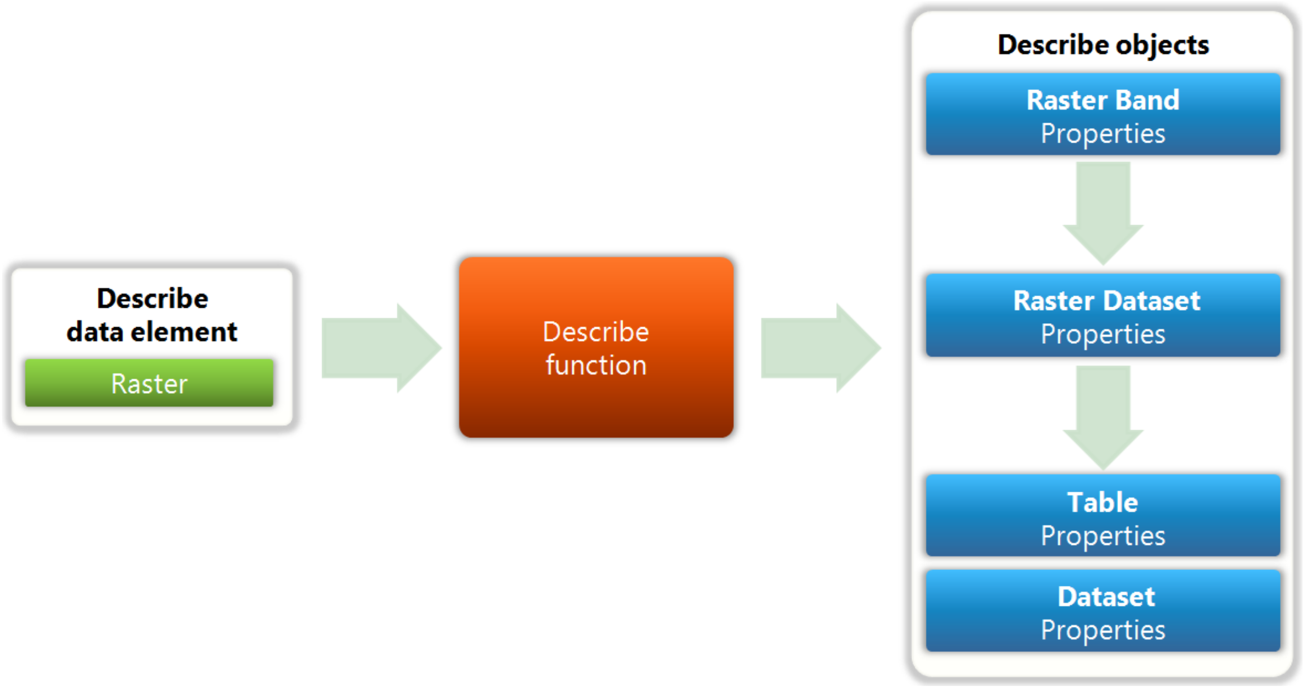
Property	Data Type
hasOID	Boolean
OIDFieldName	String
fields	Python List
indexes	Python List

Describe object

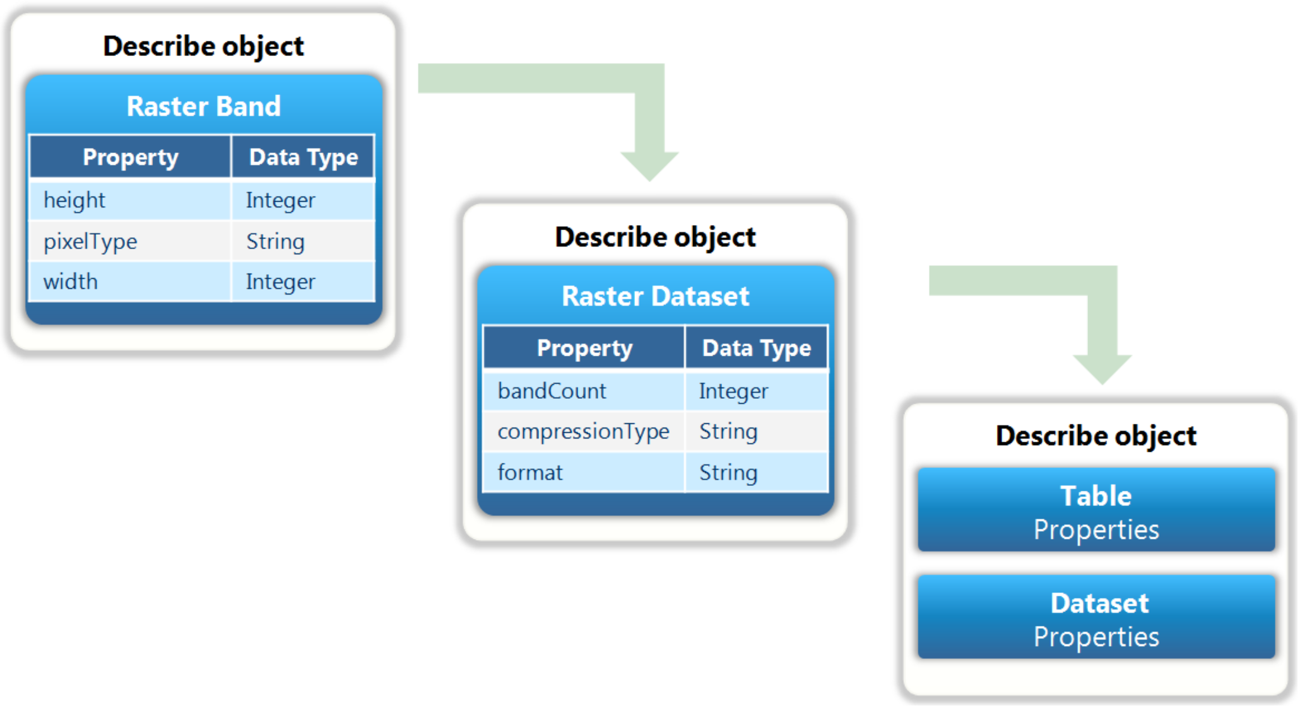
Dataset

Property	Data Type
canVersion	Boolean
datasetType	String
DSID	Integer
extent	Extent
isVersioned	Boolean
MExtent	String
spatialReference	Object
ZExtent	String

Describing a raster



Raster properties



Raster Band

Property	Data Type
height	Integer
pixelType	String
width	Integer

Raster Dataset

Property	Data Type
bandCount	Integer
compressionType	String
format	String

Activity: Describe data

The instructor will show three incomplete scripts. A copy of each is located here, so that you can fill in the missing code. Solutions are provided at the end of the lesson.

Fill in the blanks to complete the following scripts.

1. Describe a feature class.

```
# This script describes a feature class from the SanDiego.gdb
# geodatabase.

import arcpy

arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"
dscFC = arcpy.Describe("Climate")

# Fill in the blanks

print "Shape Type: " + dscFC.__________

print "Feature Type: " + dscFC.____________

print "Extent: " + __________________
```

2. Describe a dataset.

```
# This script describes a feature dataset from the Redlands.gdb
# geodatabase.

import arcpy

arcpy.env.workspace = "C:/Student/PYTH/Database/Redlands.gdb"
dscDS = arcpy.Describe("Census")

# Fill in the blank

if dscDS.____________ == "FeatureDataset":

    print "Spatial Reference: " + ____________________________

print "File Base Name: " + dscDS._________
```

3. Describe a raster.

```
# This script describes a raster from the Tahoe folder

import arcpy

arcpy.env.workspace = "C:/Student/PYTH/Database/Tahoe"
dscRS = arcpy.Describe("Emer/erelev")

# Fill in the blank

print "Number of bands: " + _____________________

if dscRS._______ == "GRID":

...

if dscRS.datasetType == "______________":...

if dscRS.________________ == "JPEG2000":

...
```

Code samples for describing data

Access the children property of a describe object

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Describe_datasetchildren.py
# Describe all the featureclasses in a database through
# the children property of the describe object.
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

import arcpy

# Variable reference for the workspace to describe
work = r"C:\Student\PYTH\Database\SanDiego.gdb"

# Describe the referenced workspace and generate a describe object.
desc = arcpy.Describe(work)

# Access the describe objects of the featureclasses it contains.
# This can be done by using the children property of the describe object
workItems = desc.children

# Create an empty dictionary to store the key::Value pairs
dictResult = {}

# The children property returns a python list of describe objects
# Use a loop to access these objects and add the name and shapetype
# as a key::Value pair to the dictionary
for item in workItems:
    if item.dataType == "FeatureClass":
        print item.name + ": " + item.shapeType
        dictResult[item.name] = item.shapeType
```

Determine if a feature class path is UNC or local

```
#~~~~~~~UNC or Local access~~~~~~~
# Describe_localorUNC.py
# This script will determine if the path to a feature class
# is UNC or mapped, or is on a local drive.
#~~~~~~~~~~~~~~~~~~~~~~~~~~~

import arcpy

# Variable reference for the featureclass
fc = r"C:\Student\PYTH\Database\Corvallis.gdb\Parcel"

# Describe the referenced featureclass and generate a describe object.
# Print some of the properties accessed through that object.
descFC = arcpy.Describe(fc)
print "Base Name: " + descFC.baseName
print "Catalog Path: " + descFC.catalogPath
print "Data Type: " + descFC.datatype

# Determine if the Feature class is being accessed
# through a mapped or UNC path
if descFC.catalogPath[0] == "\\":
    print "UNC path found"
else:
    print "Not a UNC path"
```

Exercise 4: Describe data

Estimated time: 30 minutes

The Describe function returns a Describe object that contains properties of the input data. These properties can be used to report some descriptive information of the input data, logically control the flow of a script or provide a parameter to a geoprocessing tool.

In this exercise, you will:

- Describe a feature class
- Describe a raster dataset

Step 1: Describe a feature class

In this step, you will open a new script, import the ArcPy site package, describe a feature class, and print the feature class extent and shape type to the Interactive Window.

1. What ArcGIS Desktop Help topic might you use to help write this script?

____

____

- ❑ In PythonWin, create a new Python script.
- ❑ Save the script as **DescribeFC.py** to C:\Student\PYTH\Exercise04.
- ❑ Using your skills and available resources, write code to:
  - Describe the C:\Student\PYTH\Database\Corvallis.gdb\Railroad feature class.
  - Print the shape type of the Railroad feature class to the Interactive Window.
  - Print the extent of the Railroad feature class to the Interactive Window. (*Hint:* Remember that the extent property returns an object that you access from the Dataset properties.)
- ❑ Run the script.

The shape type and extent of the Railroad feature class are printed to the Interactive Window.

2. What is the geometry shape type for the Railroad feature class?

____

- ❑ Close the DescribeFC.py script.

Step 2: Describe and clip a raster dataset

In this step, you will clip a raster dataset using the extent of another raster dataset.

For reference, here is the syntax for the Raster Clip tool:

```
Clip_management (in_raster, rectangle, out_raster,
                   {in_template_dataset}, {nodata_value}, {clipping_geometry})
```

One way to get the clipping extent rectangle (a required parameter) is to refer to the raster dataset properties in ArcCatalog, and manually type in the minimum and maximum x,y values for your rectangle parameter.

Another way is to use a Describe object in your script. In this step, you will use a Describe object to return the extent of a raster dataset. Then you will use the returned extent to clip another raster dataset.

- ❑ Create a new Python script.
- ❑ Save the script as **ClipRDExtent.py** to C:\Student\PYTH\Exercise04.
- ❑ Using your skills and available resources, write code to:
  - Describe the C:\Student\PYTH\Database\Tahoe\Emer\erelev raster dataset.
  - Print the extent of erelev to the Interactive Window.
- ❑ Run the script.

Now you will verify the extent values that the `Describe` function returned for the erelev raster dataset.

- ❑ Open ArcCatalog and navigate to the C:\Student\PYTH\Database\Tahoe\Emer folder.
- ❑ Right-click erelev and choose Properties.

3. Scroll down to the Extent properties and record the values below.

Top		`YMax`
Left		`XMin`
Right		`XMax`
Bottom		`YMin`

❑ Close ArcCatalog and return to PythonWin.

4. Is the extent returned from the `Describe` function the same as the extent that you obtained from ArcCatalog?

___

Now you are ready to run the Clip tool in your script.

❑ Clip the \All\arowner raster with the extent of \Emer\erelev to create a new \Emer\erowner raster.

- For the Raster Clip tool's second argument, use the extent of erelev.
  (*Hint:* Remember that the Describe function returns an object. The Clip tool will need the string representation of the extent property.)

❑ Run the script.

❑ Verify your clip results:

- Start ArcMap, and open a blank map.
- Add the following rasters: \All\arowner, \Emer\erelev, and \Emer\erowner.
- Compare the rasters to verify the results of your clip operation.
- Close ArcMap and do not save the map.

❑ Close the ClipRDExtent.py script.

Lesson review

1. When describing a feature class, what are some of the properties returned?

2. What is returned for an Extent property of a Dataset?

3. Give an example of how you could use a Describe object in a geoprocessing tool.

Answers to Lesson 4 questions

Activity: Describe data

1. Describe a feature class.

```
# This script describes a feature class from the SanDiego.gdb
# geodatabase.

import arcpy

arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"
dscFC = arcpy.Describe("Climate")

# Fill in the blanks
print "Shape Type: " + dscFC.shapeType
print "Feature Type: " + dscFC.featureType
print "Extent: " + str(dscFC.extent)
```

2. Describe a dataset.

```
# This script describes a feature dataset from the Redlands.gdb
# geodatabase.

import arcpy

arcpy.env.workspace = "C:/Student/PYTH/Database/Redlands.gdb"
dscDS = arcpy.Describe("Census")

# Fill in the blank
if dscDS.datasetType == "FeatureDataset":
    print "Spatial Reference: " + str(dscDS.spatialReference)
print "File Base Name: " + dscDS.baseName
```

3. Describe a raster.

```
# This script describes a raster from the Tahoe folder

import arcpy

arcpy.env.workspace = "C:/Student/PYTH/Database/Tahoe"
dscRS = arcpy.Describe("Emer/erelev")

# Fill in the blank
print "Number of bands: " + str(dscRS.bandCount)
if dscRS.format == "GRID":
...
if dscRS.datasetType == "RasterDataset":...
if dscRS.compressionType == "JPEG2000":
...
```

Exercise 4: Describe data

1. What ArcGIS Desktop Help topic might you use to help write this script?

   Professional Library > Geoprocessing > The ArcPy site package > Functions > Describing data > Describe properties > FeatureClass properties

2. What is the geometry shape type for the Railroad feature class?

   Polyline

3. Scroll down to the Extent properties and record the values below.

Top	**4320842.5**	`YMax`
Left	**748982.6875**	`XMin`
Right	**760262.6875**	`XMax`
Bottom	**4306622.5**	`YMin`

4. Is the extent returned from the `Describe` function the same as the extent that you obtained from ArcCatalog?

   Yes.

Lesson review

1. When describing a feature class, what are some of the properties returned?

   - **featureType**
   - **shapeType**
   - **shapeFieldName**
   - **hasM**
   - **hasZ**

2. What is returned for an Extent property of a Dataset?

   An Extent object, which has properties for X-Min, Y-Min, X-Max, Y-Max, as well as other properties.

3. Give an example of how you could use a Describe object in a geoprocessing tool.

   Use the Extent object to clip a raster or feature class

Exercise solution

DescribeFC.py

```
# Author: ESRI
# Date:
# Purpose: To describe a feature class.

# Import the arcpy site package
import arcpy

# Describe the Railroads feature class.
dscFC = arcpy.Describe("C:\\Student\\PYTH\\Database\\Corvallis.gdb\\Railroad")

# Print the shape type and extent of the Railroad feature class
# to the Interactive Window.
print dscFC.shapeType
print dscFC.extent

#Alternate solution to printing the Extent properties
#print dscFC.extent.XMin
#print dscFC.extent.YMin
#print dscFC.extent.XMax
#print dscFC.extent.YMax
```

ClipRDExtent.py

```
# Author: ESRI
# Date:
# Purpose: Clip a raster by the extent of another raster
#                using a describe object

# Import the arcpy site package
import arcpy

# Set the workspace environment value
arcpy.env.workspace = "C:\\Student\\PYTH\\Database\\Tahoe"

# Describe the Tahoe\Emer\erelev raster
dscRS = arcpy.Describe("Emer\\erelev")

# Print the extent of the erelev raster dataset
# to the Interactive Window.
print dscRS.extent
```

```
# Clip the All\arowner with Emer\erelev to create Emer\erowner
arcpy.Clip_management("All\\arowner", str(dscRS.extent), "Emer\\erowner")
```

5 Automating scripts with Python lists

Introduction

One of the primary tasks in scripting is automating the processing of data with a list of data. The ArcPy site package has many list functions that are built to return Python lists for different types of data. In this lesson, you will explore these ArcPy list functions.

Learning objectives

After completing this lesson, you will be able to:

- Determine the proper List function to use
- Perform geoprocessing of data using Python lists

The List functions

The ArcPy site package provides a number of List functions that return a list of values. The List functions can provide you with a Python list of feature class names in a geodatabase, shapefile names in a directory, table names in a geodatabase, fields in a feature class or table, and many additional lists.

The scripts that you write can iterate through each item in the Python list and perform defined tasks. You will take a look at these List functions and see how to use them in automation workflows.

Explore the ArcGIS Desktop Help:

- Professional Library >
- Geoprocessing >
- Geoprocessing with Python >
- Working with sets of data in Python >
- Listing data

Click the List function links to help you answer the following questions:

1. The `ListFeatureClasses` function returns a Python list. What data type is the returned value? What do the values contain?

   ____

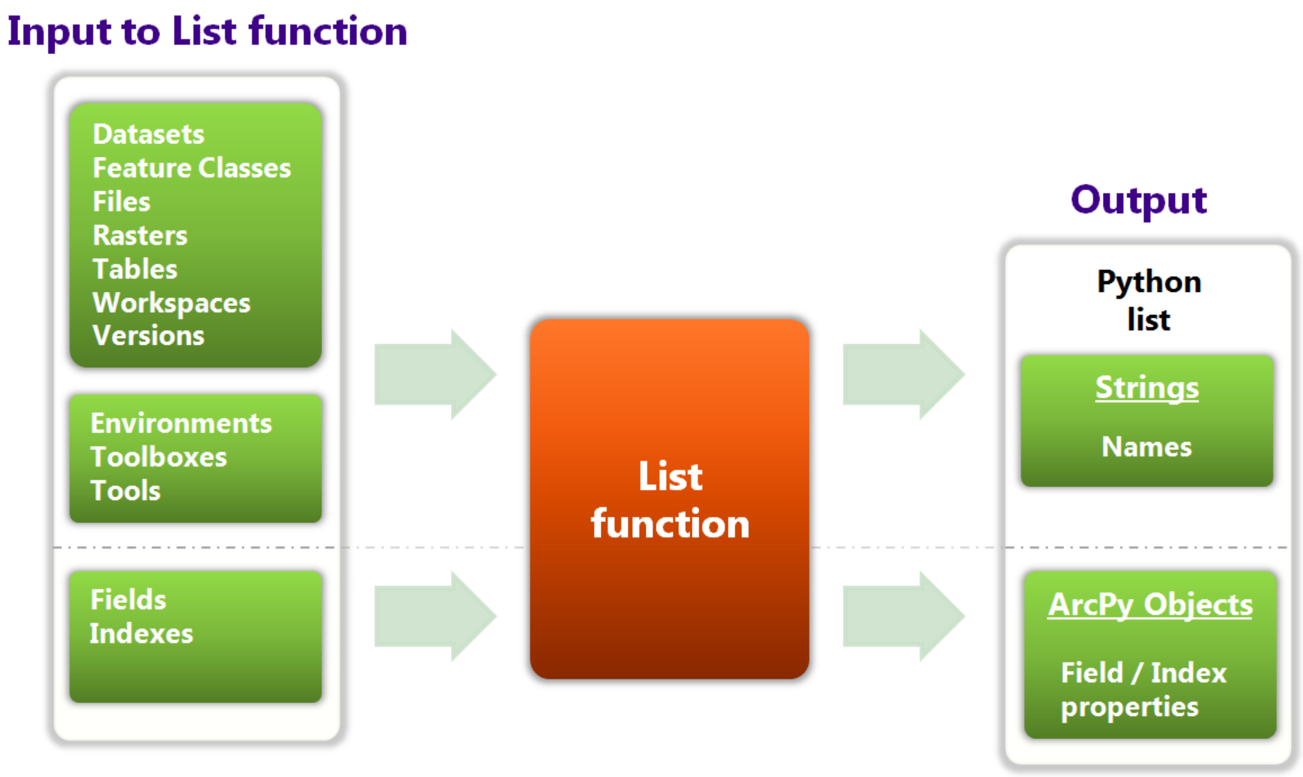
2. The `ListFields` function returns a Python list that contains `Field` objects. What are some of the field object properties that you can access?

   ____

   ____

3. Referring to the code sample for the `ListFields` function, how do you iterate through the Python list that the function returns?

   ____


Input to List function
Datasets
Feature Classes
Files
Rasters
Tables
Workspaces
Versions
Environments
Toolboxes
Tools
Fields
Indexes
List function
Output
Python list
Strings
Names
ArcPy Objects
Field / Index properties

Activity: Create Python lists

The instructor will show three incomplete scripts. A copy of each is located here, so that you can fill in the missing code. Solutions are provided at the end of the lesson.

Fill in the blanks to complete the following scripts.

1. List datasets.

```
# This script creates a Python List of feature datasets
# from the World.gdb geodatabase. A for loop is used to
# iterate through the Python List and print the dataset
# names to the Interactive Window.

import arcpy

arcpy.env.workspace = "C:\\Student\\PYTH\\Database\\World.gdb"

# Fill in the blanks

dsList = arcpy.List_________("*")


for ds in _______:

    ______ ds
```

2. List field names and type.

```
# This script creates a Python List of field names
# from the MajorAttractions feature class in the SanDiego.gdb
# geodatabase.

import arcpy

arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"

# Fill in the blanks

fldList = arcpy.ListFields("_________________")

for fld in fldList:

    print ____._____

    print ____._____
```

3. List rasters and build pyramids.

```
# This script creates a Python List of rasters in the Tahoe
# folder that start with "e", then builds pyramids on each
# raster.

import arcpy

arcpy.env.workspace = "C:/Student/PYTH/Database/Tahoe/Emer"

# Fill in the blanks

rdList = arcpy.____________("___")

for ras in rdList:

    arcpy.BuildPyramids_management(____)
```

Iterating through lists

All the ArcPy List functions return a Python list of items. To process the items in the Python list, you can iterate through the list using a Python For.. loop.

The following sample script illustrates how to iterate through a Python list of feature class names.

Workflow

1. Import `arcpy`
2. Set the current workspace as the location to look for the feature classes
3. Call the `ListFeatureClasses` function and assign the return to a variable
4. Iterate with a `for` loop through the variable, which contains a Python list
5. For each item in the list, print the item to the Interactive Window

```
# This script creates a Python List of feature classes
# from the SanDiego.gdb geodatabase. A for loop is used
# to iterate through the Python List and print the
# feature class names to the Interactive Window.

# Import ArcPy site package and set the current workspace
import arcpy

arcpy.env.workspace = "C:\\Student\\PYTH\\Database\\SanDiego.gdb"

# Create a Python List of feature class names.
# You can use a wild card for the names or just leave the
# argument blank. fcList will be assigned the Python list
# returned from the ListFeatureClasses function.

fcList = arcpy.ListFeatureClasses("*")

# Iterate through the Python List with a for..in loop
# and print to the Interactive Window.

for fc in fcList:
    print "Feature class name: " + fc
```

Code samples for listing data

Listing feature classes

```
# Name: ListFeatureClasses.py
# Date: <today>
# Purpose:  List all of the feature classes in the current workspace
#           to the PythonWin Interactive Window

# Import ArcPy site package
import arcpy

# Set the current workspace
arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"

# Create a Python list of Feature Class Names
# No arguments means all feature class names returned.
lstFC = arcpy.ListFeatureClasses()

# Iterate through the Python list and print to Interactive Window
# using a for..in loop
for fc in lstFC:
    print "Feature Class Name: " + fc
```

Listing field properties

```
# Name: ListClimateFields.py
# Date: <today>
# Purpose:  List all of the fields in the Climate feature class
#           to the PythonWin Interactive Window

# Import ArcPy site package
import arcpy

# Set the current workspace
arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"

# Create a Python list of Fields in the Climate feature class
# Required argument is the table/feature class
lstFields = arcpy.ListFields("Climate")
```

```
# Iterate through the Python list using a for..in loop
# Each item returned from the Python list is a Field object.
# Print the Field name and type to the Interactive Window
for field in lstFields:
    print "Field: " + field.name + " has a field type of: " + field.type
```

Listing geodatabases in a workspace

```
# Name: ListGeodatabases.py
# Date: <today>
# Purpose:  List all of the file geodatabases in the current workspace
#           to the PythonWin Interactive Window

# Import ArcPy site package
import arcpy

# Set the current workspace
arcpy.env.workspace = "C:/Student/PYTH/Database"

# Create a Python list of File Geodatabases
# Required argument is a wild card for the name
# and the workspace type.
lstWorkspace = arcpy.ListWorkspaces("*", "FileGDB")

# Iterate through the Python list and print to Interactive Window
# using a for..in loop
for fileGDB in lstWorkspace:
    print "File geodatabase Name: " + fileGDB
```

List all text files in a workspace

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ListTextFiles.py
# List all textfiles in specified location
# Load contents of txt file into a Python dictionary
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Import arcpy site package and set workspace
import arcpy
arcpy.env.workspace = r"C:\student\Pyth\Database"
```

```
# Create Python list of text file names
listTxt = arcpy.ListFiles("*.txt")

# Create Python dictionary to contain text file contents
txtDict = {}

# If text file(s) found in specified folder
# loop through each text file and store contents in
# the Python dictionary
if len(listTxt) > 0:
    for txtFile in listTxt:
        txtFilePath = arcpy.env.workspace + "\\" + txtFile
        openFile = open(txtFilePath, 'r')
        lines = len(openFile.readlines())
        txtDict[txtFile] = lines
```

List all feature classes using Python os.walk

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ListFeatureClasses_OSwalk.py
# Starting at a specified workspace location,
# list all feature classes found at that location
# and in any sub-folder using the os module.
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

import arcpy, os

startLocation = r"C:\Student\PYTH"

foundFCS = {}

for root, dirs, files in os.walk(startLocation):
    #Set the Workspace to the current location
    arcpy.env.workspace = root

    #List the featureclasses in the Current Workspace
    lfc = arcpy.ListFeatureClasses()
    totalFCS = len(lfc)

    #Add the result to the dictionary
    foundFCS[root] = totalFCS
```

Exercise 5: Working with lists

Estimated time: 30 minutes

The ArcPy List functions can catalog available data filtered by the type of data for which you are looking and return a Python list of values. These values may be items such as feature class names, raster names, shapefiles in a folder, or a list of fields in a table.

In this exercise, you will write scripts to:

- List the geodatabases in a folder
- List all the fields in a feature class
- Delete all the specified rasters in a folder

Step 1: List all the file geodatabases in a folder

In this step, you will open a new script, import the ArcPy site package, list all the geodatabases in the C:\Student\PYTH\Database folder, and print the name of each geodatabase to the Interactive Window.

❑ Create a new Python script named **ListGDB.py** and save it to C:\Student\PYTH\Exercise05.

❑ List all the file geodatabases in the C:\Student\PYTH\Database folder.

> **Note:** Refer to the *Listing Data* help topic to determine the correct List function to use. (ArcGIS Desktop Help > Professional Library > Geoprocessing > Geoprocessing with Python > Working with sets of data in Python)

❑ Iterate through the returned Python list and print the name of each geodatabase to the Interactive Window.

❑ Run the script.

1. What file geodatabases are stored in the Database folder?

____

❑ Close the ListGDB.py script.

Step 2: List all the fields in a feature class

In this step, you will open a new script, import the ArcPy site package, list all the fields in the C:\Student\PYTH\Database\SanDiego.gdb\MajorAttractions feature class, and print the name, field type, and field length of each field to the Interactive Window.

- ❑ Create a new Python script named **ListFields.py** and save it to C:\Student\PYTH\Exercise05.
- ❑ List all the fields in the C:\Student\PYTH\Database\SanDiego.gdb\MajorAttractions feature class.
- ❑ Iterate through the returned Python list and print the field name, field type, and field length of each field to the Interactive Window.
  - To concatenate a number to a string, use the `str` function.
  - Remember that each field returned from the List function is an object.
- ❑ Run the script.

2. Refer to the Interactive Window to complete the following table:

Field name	Field type	Field length
OBJECTID		
ESTAB		
ZIP		
Shape		

- ❑ Close the ListFields.py script.

Step 3: Delete raster datasets in a folder

In this step, you will write code to delete raster datasets in a folder.

- ❑ Create a new Python script named **DeleteRasters.py** and save it to C:\Student\PYTH\Exercise05.
- ❑ List the raster datasets with names that start with the letter "e" that are stored in the C:\Student\PYTH\Database\Tahoe\Emer folder.

- ❑ Iterate through the returned Python list and delete each raster.
  (*Hint:* There is a geoprocessing tool in the Data Management toolbox that will delete data.)
- ❑ Run the script.
- ❑ Open ArcCatalog to verify that the Emer folder does not contain raster datasets that begin with the letter "e." (You may need to refresh the folder if you already had ArcCatalog open.)
- ❑ Close the DeleteRasters.py script.

Lesson review

1. What do all List functions on the geoprocessor return?

2. To loop through a Python list that gets returned from a List function, which of the following would you use?
   a. A for loop.
   b. A range loop.
   c. A while loop.

3. Write a short script that does the following:
   - Lists all the feature datasets in ..\Student\PYTH\Database\World.gdb.
   - Prints the names of the feature datasets to the Interactive Window.

Answers to Lesson 5 questions

The List functions

1. The `ListFeatureClasses` function returns a Python list. What data type is the returned value? What do the values contain?

   A `String` value is returned that contains the names of the feature classes.

2. The `ListFields` function returns a Python list that contains `Field` objects. What are some of the field object properties that you can access?

   Some field object properties include `name`, `type`, `length`, `scale`, `aliasName`, etc.

3. Referring to the code sample for the `ListFields` function, how do you iterate through the Python list that the function returns?

   Use a For.. loop.

Activity: Create Python lists

1. List datasets.

```
# This script creates a Python List of feature datasets
# from the World.gdb geodatabase. A for loop is used to
# iterate through the Python List and print the dataset
# names to the Interactive Window.

import arcpy

arcpy.env.workspace = "C:\\Student\\PYTH\\Database\\World.gdb"

# Fill in the blanks
dsList = arcpy.ListDatasets("*")

for ds in dsList:
    print ds
```

2. List field names and type.

```
# This script creates a Python List of field names
# from the MajorAttractions feature class in the SanDiego.gdb
# geodatabase.

import arcpy

arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"

# Fill in the blanks
fldList = arcpy.ListFields("MajorAttractions")

for fld in fldList:
    print fld.name
    print fld.type
```

3. List rasters and build pyramids.

```
# This script creates a Python List of rasters in the Tahoe
# folder that start with "e", then builds pyramids on each
# raster.

import arcpy

arcpy.env.workspace = "C:/Student/PYTH/Database/Tahoe/Emer"

# Fill in the blanks
rdList = arcpy.ListRasters("e*")

for ras in rdList:
    arcpy.BuildPyramids_management(ras)
```

Exercise 5: Working with lists

1. What file geodatabases are stored in the Database folder?

   Corvallis.gdb, Redlands.gdb, SanDiego.gdb, World.gdb

2. Refer to the Interactive Window to complete the following table:

Field name	Field type	Field length
OBJECTID	**OID**	**4**
ESTAB	**Integer**	**4**
ZIP	**String**	**5**
Shape	**Geometry**	**0**

Lesson review

1. What do all List functions on the geoprocessor return?

   A Python List of files that contain either string values or objects.

2. To loop through a Python list that gets returned from a List function, which of the following would you use?

   a. A for loop.

3. Write a short script that does the following:
   - Lists all the feature datasets in ..\Student\PYTH\Database\World.gdb.
   - Prints the names of the feature datasets to the Interactive Window.

```
# Import ArcPy site package
import arcpy

#Set workspace environment
arcpy.env.workspace = "C:/Student/PYTH/Database/World.gdb"

#Obtain a list of feature datasets in the World.dgb file geodatabase
lstFDS = arcpy.ListDatasets("*", "FeatureDataset")

print "Feature datasets in " + arcpy.env.workspace

# Iterate through the list and print to Interactive Window
for featureDataset in lstFDS:
    print "\t" + featureDataset #use a '\t' to print a tab
```

Exercise solution

ListGDB.py

```
# Author: ESRI
# Date:
# Purpose: Lists all the geodatabases in a folder.

# Import the ArcPy site package
import arcpy

# Set the workspace.
arcpy.env.workspace = "C:\\Student\\PYTH\\Database"

# List all of the geodatabases in the Database folder.
# gdbList is a Python List returned from the ListWorkspaces function.
gdbList = arcpy.ListWorkspaces("*", "FileGDB")

# Iterate through all the geodatabases and print the name of each
# geodatabase to the Interactive Window.
for gdb in gdbList:
    print gdb
```

ListFields.py

```
# Author: ESRI
# Date:
# Purpose: Lists all the fields in a feature class.

# Import the ArcPy site package
import arcpy

# Set the workspace.
arcpy.env.workspace = "C:\\Student\\PYTH\\Database\\SanDiego.gdb"

# List all of the fields in the MajorAttractions folder.
# fldList is a Python List returned from the ListFields function.
fldList = arcpy.ListFields("MajorAttractions")

# Iterate through the fields and print the name of each
# field to the Interactive Window.
for fld in fldList:
    print fld.name + " is a " + fld.type + " field with a length of " + str(fld.length)
```

DeleteRasters.py

```
# Author: ESRI
# Date:
# Purpose: Lists all the rasters that start with 'e'
# in the Tahoe\Emer folder.

# Import the ArcPy site package
import arcpy

# Set the workspace.
arcpy.env.workspace = "C:\\Student\\PYTH\\Database\\Tahoe\\Emer"

# List all the rasters that start with 'e' in the current workspace
# rdList is a Python List returned from the ListRasters function.
rdList = arcpy.ListRasters("e*")

#Iterate through the list and call the Delete tool passing in
# the name of the raster to delete.
for ras in rdList:
    arcpy.Delete_management(ras)
```

6 Creating and updating data with Cursor objects

Introduction

A cursor is a data object that can be used to iterate over a set of rows in a table or feature class to read field values, update field values or insert new rows. By using cursors, you can quickly process subsets of data for mass updates of field values or generate reports on existing field values. You can use a cursor to read a text file and create new rows in the table.

Learning objectives

After completing this lesson, you will be able to:

- Describe the three forms of cursors
- Read values from fields using a cursor
- Add a new field and update field values

Cursor objects

There are many reasons to work with cursors. The most typical workflows that use cursors use them to mass update field values (such as when a new field is added), to summarize and report on field values, and to mass insert new rows into a table or feature class.

In this lesson, you will write Python scripts that use cursors to read values from fields and to update a field value by concatenating the values from other fields. One of the first steps to writing a script is to gain an understanding of the resources that the script needs to perform the required tasks.

In the ArcGIS Desktop Help, navigate to the following topic:

- Professional Library >
- Geoprocessing >
- The ArcPy site package >
- Classes >
- Cursor

1. What is a cursor?

____

2. What are the three forms of a cursor?

____

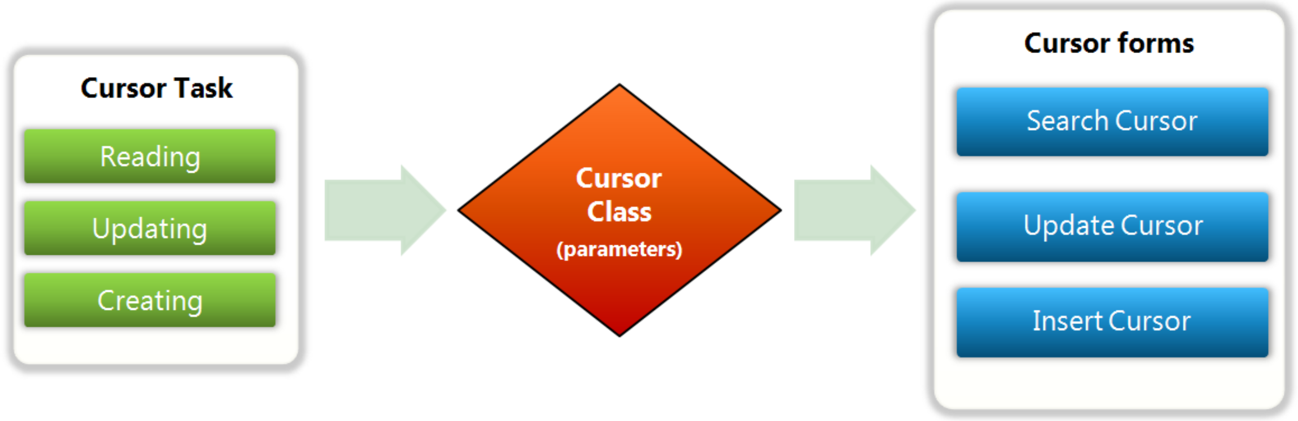
In the Help, expand Functions > Cursors and review the three cursor topics.

3. Which cursor function can be used to create a cursor object that retrieves rows?

____

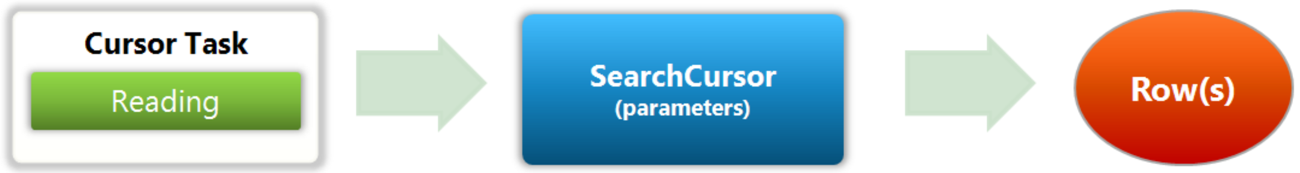
4. If your script places new values in fields for existing rows, which cursor function would you use?

____



Cursor functions

SearchCursor function



```
#Set current workspace
arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"

#Create cursor on MajorAttractions feature class
cur = arcpy.SearchCursor("MajorAttractions")

# Iterate through the rows in the cursor
# Print the name and Address of each Major Attraction
for row in cur:
    print row.Name
    print row.Addr
del cur, row
```

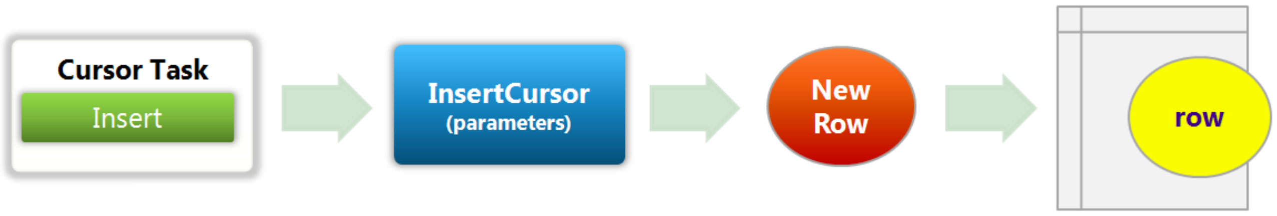
UpdateCursor function

Cursor Task
Updating
UpdateCursor
(parameters)
Row(s)

```
#Set current workspace
arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"

#Create cursor on MajorAttractions feature class
cur = arcpy.UpdateCursor("MajorAttractions",
      '"NAME" = \'San Diego Zoo\'')

# Iterate through the rows in the cursor and update the address
for row in cur:
    row.Addr = "1900 ZOO PLACE"
    cur.updateRow(row)
del cur, row
```

Notes

InsertCursor function



```
#Set current workspace
arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"

#Create cursor on MajorAttractions feature class
cur = arcpy.InsertCursor("MajorAttractions")

# Create new row, update fields and insert row.
row = cur.newRow()
row.Name = "BLACK MOUNTAIN PARK"
row.Addr = "12115 BLACK MOUNTAIN RD"
cur.insertRow(row)
del cur, row
```

The Row object

All cursor forms return a Row object. Depending on the cursor form, the Row object can access field values, set new field values, check to see if the field contains a Null value, or can set the field value to Null.

When working with the Row object, you can reference the field directly with a hard-coded field name in the form of `row.fieldname`. For example:

- `row.ADDRESS`
- `row.Acres`

The following table lists the methods on a row object.

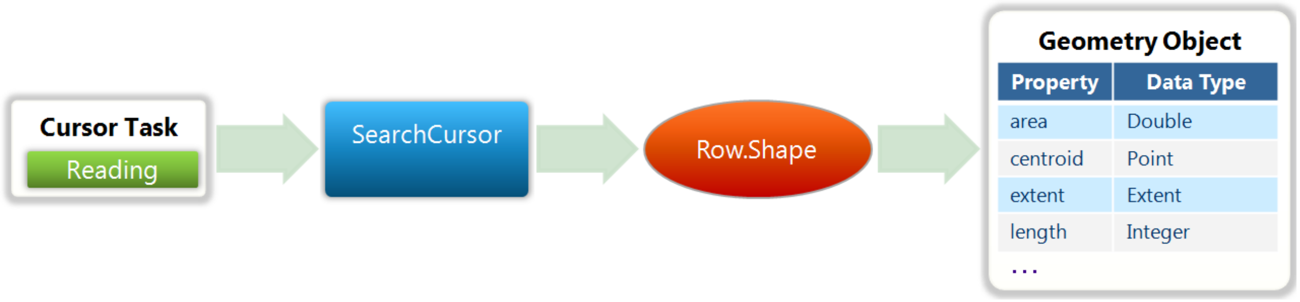
Method	Data Type	Supported by:
getValue(field_name)	Object	Search Cursor, Update Cursor
setValue(field_name, object)	Object	Update Cursor, Insert Cursor
isNull(field_name)	Boolean	Search Cursor, Update Cursor
setNull(field_name)	N/A	Update Cursor, Insert Cursor

- The `field_name` parameter is a string value referencing the field name.
- The Object data type contains the value of the field.

The `row.getValue()` and `row.setValue()` methods are provided so that variables can be substituted for field names when accessing field values or setting field values on the Row object. For example:

```
cur = arcpy.SearchCursor("MajorAttractions")
for row in cur:
    fieldName = "TYPE"
    # The next two lines are equivalent
    value = row.getValue(fieldName)
    value = row.TYPE
```

Accessing Geometry object properties



Property	Data Type
area	Double
centroid	Point
extent	Extent
length	Integer
...	

```
import arcpy

#Set current workspace
arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"

#Create cursor on Climate feature class
cur = arcpy.SearchCursor("Climate")

# Iterate through the rows in the cursor
for row in cur:
    geom = row.Shape # Returns a geometry object
    print "Polygon ID: " + str(row.ObjectID)
    print "Area: " + str(geom.area) # row.Shape.area
    print "Centroid: " + str(geom.centroid.X) + ", " + \
          str(geom.centroid.Y) + "\n"
```

Code samples using cursors

A short collection of Python scripts for working with cursors

Use a SpatialReference in the SearchCursor function

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# cursor_spatialreference.py
# This script passes a spatial reference object
# to the SearchCursor function
# The spatial reference of each row returned by the
# SearchCursor will be set to the cursor spatial reference
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

import arcpy

# Provide a variable reference for the input featureclass
fc = r"C:\Student\PYTH\Database\SanDiego.gdb\Zipcodes"

# Reference the projected coordinate system
utmPRJ = r"C:\Program Files\ArcGIS\Desktop10.0" \
         "\Coordinate Systems\Projected Coordinate Systems" \
         "\UTM\NAD 1983\NAD 1983 UTM Zone 11N.prj"

# Build a spatial reference object, passing the .prj file
spatRef = arcpy.SpatialReference(utmPRJ)

# Create a SearchCursor using the spatial reference object
# to change geometry properties to meters
src = arcpy.SearchCursor(fc,"", spatRef)

# Loop through the results and print the area in meters
for zip in src:
    print zip.ZIP
    print "The area in meters is: " + str(zip.shape.area)

# Release the cursor
del src
```

Use a WHERE clause in the SearchCursor function

```
#~~~~~~~~~~~~~~~~~~~~~~
# cursor_whereClause.py
# This script shows how to pass a SQL here clause
# to a Search Cursor and store the results in a dictionary.
#~~~~~~~~~~~~~~~~~~~~~~

import arcpy

# Variable reference for the featureclass
fc = r"C:\Student\PYTH\Database\SanDiego.gdb\MajorAttractions"

# Variable reference for the SQL where clause to pass to the cursor.
# When applied limits the records returned by the cursor
whereClause = "EMP > 500"

# Create a Search Cursor passing the
# reference featureclass and the whereclause
src = arcpy.SearchCursor(fc, whereClause)

# Store the results in a dictionary to lookup later
# This technique allows you to lookup the place, and return the
# employee value.
# eg. ShamuVal = majorAttractionLookup["SAN DIEGO ZOO"]
majorAttractionLookup = {}

# Loop through the results and store them in the python dictionary
for place in src:
    print place.getValue("name")
    majorAttractionLookup[place.getValue("name")] = place.EMP

# Release the cursor
del src

# Print the results of applying the whereclause
print "There are " + str(len(majorAttractionLookup)) + \
      " Attractions that employ over 500 people"
```

Exercise 6: Use the SearchCursor and UpdateCursor functions

Estimated time: 40 minutes

In this exercise, you will use the SearchCursor function to read field values from a feature class. Then you will use the UpdateCursor function to place values in a field based on values from two other fields.

In this exercise, you will:

- Access field values from a feature class
- Add and update a field
- Optionally, check for a field

Step 1: Access field values

In this step, you will open a new script, import the ArcPy site package, create the cursor object, then print the name, address, city, and zip of each feature in the MajorAttractions feature class to the Interactive Window.

- ❑ Create a new Python script named **Attractions.py** and save it to C:\Student\PYTH\Exercise06.

1. What ArcPy List function can you use to determine the field names of a feature class or table?

____

2. When iterating through the Python List that this function returns, how can you determine the name of each field?

____

❑ Using the SearchCursor function:
  - Print the name, address, city, and zip code for each feature in the C:\Student\PYTH\Database\SanDiego.gdb\MajorAttractions feature class to the Interactive Window.(*Hint*: Use your answers to the previous questions.)
  - Format the output as follows in a three-line address style:

```
FOUR POINTS HOTEL
8110 AERO DR
SAN DIEGO, CA 92123
```

❑ Run the script.

Addresses for several attractions print to the Interactive Window.

❑ Scroll to the end of the list of addresses.

3. What is the address for the Hilton Crystal Bay Hotel?

____

____

❑ Close the Attractions.py script.

Step 2: Add and update a field

In this step, you will add a field to a feature class and populate that field with values from two other fields.

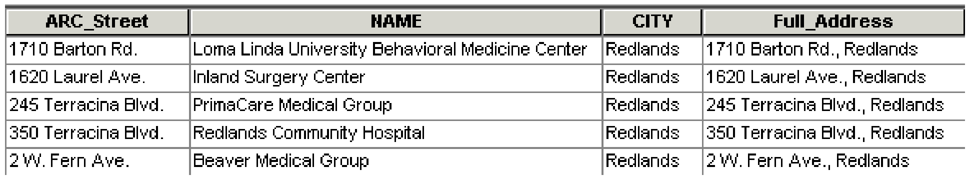
❑ Create a new Python script named **UpdateField.py** and save it to C:\Student\PYTH\Exercise06.

❑ Use the AddField geoprocessing tool to add a Text field called Full_Address to the C:\Student\PYTH\Database\Redlands.gdb\Hospitals feature class.

4. Which type of cursor will allow you to write new attribute values to existing rows in a feature class?

____

- ❑ Populate your new field with values from the ARC_Street and CITY fields. Be sure to include a comma and space when concatenating the field values.
- ❑ Because cursors place locks on the dataset that is being updated, add code to the bottom of your script to delete the cursor.
- ❑ Run the script.
- ❑ Open ArcCatalog and view the updated table for the Hospitals feature class.



ARC_Street	NAME	CITY	Full_Address
1710 Barton Rd.	Loma Linda University Behavioral Medicine Center	Redlands	1710 Barton Rd., Redlands
1620 Laurel Ave.	Inland Surgery Center	Redlands	1620 Laurel Ave., Redlands
245 Terracina Blvd.	PrimaCare Medical Group	Redlands	245 Terracina Blvd., Redlands
350 Terracina Blvd.	Redlands Community Hospital	Redlands	350 Terracina Blvd., Redlands
2 W. Fern Ave.	Beaver Medical Group	Redlands	2 W. Fern Ave., Redlands

The Full_Address field that you added to the Hospitals feature class contains the values from both the ARC_Street and CITY fields, separated by a comma and a space.

- ❑ Close ArcCatalog.
- ❑ If you have time, complete the optional step; otherwise, close the UpdateField.py script.

Step 3: (Optional) Check for a field

Any time that your script modifies the schema of a dataset, you should first check to see if that change has already been made so that your script does not crash.

In this optional step, you will add code to your script to first check for a field with the same name and add the new field to the feature class if it does not already exist.

- ❑ In PythonWin, save the UpdateField.py script as C:\Student\PYTH\Exercise06\CheckAndUpdateField.py.

5. Which Python built-in function will return the number of items in a Python list?

____

- ❑ Above the code that adds the new field, modify the script to do the following:
  - Check if the Full_Address field exists.
  - If the field does not exist, add it to the Hospitals feature class.
  - If the field already exists, print a message to the Interactive Window.
- ❑ Check for syntax errors and run the script.

You created the field in the previous step, so the Interactive Window prints your message which indicates that the field already exists.

In your script, you used the hard-coded Hospitals value for the feature class and the Full_Address value for the new field. When a hard-coded value is used many times throughout a script, a best practice is to store that value in a variable.

Using variables in the place of hard-coded values can make your script easier to modify. If you decide to reuse the script on a different feature class and/or field, only the variable values would need to be changed.

- ❑ Above the code that checks for the Full_Address field, store the following values in variables, then use the variables throughout the script:
  - Hospitals feature class
  - Full_Address field

Note: Use descriptive variable names so that your script is easier to understand and debug. The exercise solution uses the following variable names:

- `inputFC`: feature class
- `field`: field

6. When the field name is stored in a variable, instead of using the row.fieldName property to assign the field value to each row in the cursor, what method should you use?

   ____

- ❑ In your `for` loop, modify the code to use the appropriate method to assign the value to the row.
- ❑ Check for syntax errors.

- ❑ If you want to test your new script:
  - Change the value of your variable for the new field to create another new field in the Hospitals feature class.
  - Run the script.
  - Open ArcCatalog and view the table for the Hospitals feature class to see your new field (it will have the same attributes as the Full_Address field that you created earlier).
- ❑ Close ArcCatalog and close your script.

Lesson review

1. Return the maximum and minimum values of LAND_VALUE from the Corvallis Parcel feature class.
   - Write a single line of code.
   - Refer to the ArcGIS Help topic: SearchCursor.

Best practices

When working with cursors:

- Use variables instead of hard-coded values where possible. This makes your code reusable, more flexible, and easier to debug.
- When getting or updating field values for the Row object returned by the cursor, use the methods, `row.getValue()` and `row.setValue()`. This allows for variable substitution of the row field names.
- Use `del cur` and `del row` at the end of your script to remove any locks on the feature class or table. All cursors place locks on the feature class or table, which will not be removed until either the script closes or the `del` statement is used.
- Before you can use an update cursor or insert cursor on a feature class or table, the cursor must obtain exclusive access to the data; therefore, the feature class or table cannot also be open in ArcCatalog or ArcMap.

Answers to Lesson 6 questions

Cursor objects

1. What is a cursor?

   A cursor is a data object that can be used to read and update attributes.

2. What are the three forms of a cursor?

   Insert, search, and update

3. Which cursor function can be used to create a cursor object that retrieves rows?

   The SearchCursor function creates a read-only cursor object.

4. If your script places new values in fields for existing rows, which cursor function would you use?

   The UpdateCursor function creates a cursor that can update field values or delete rows.

Exercise 6: Use the SearchCursor and UpdateCursor functions

1. What ArcPy List function can you use to determine the field names of a feature class or table?

   arcpy.ListFields()

2. When iterating through the Python List that this function returns, how can you determine the name of each field?

   Use the field.name property.

3. What is the address for the Hilton Crystal Bay Hotel?

   900 F ST
   CHULA VISTA, CA 91910

4. Which type of cursor will allow you to write new attribute values to existing rows in a feature class?

   Update cursor. (A search cursor only allows you to read values and an insert cursor only allows you to add new rows.)

5. Which Python built-in function will return the number of items in a Python list?

   `len`

6. When the field name is stored in a variable, instead of using the row.fieldName property to assign the field value to each row in the cursor, what method should you use?

   row.setValue()

Lesson review

1. Return the maximum and minimum values of LAND_VALUE from the Corvallis Parcel feature class.
   - Write a single line of code.
   - Refer to the ArcGIS Help topic: SearchCursor.

```
rows = arcpy.SearchCursor("C:/Student/Data/Corvallis.gdb/Parcel",
                          "", "", "", LAND_VALUE D")
```

Exercise solution

Attractions.py

```
# Author: ESRI
# Purpose: Prints the name, address, city, state and zip of each feature
#               in the MajorAttractions feature class.

# Import the arcpy site package.
import arcpy

# Set the workspace.
arcpy.env.workspace = "C:\\Student\\PYTH\\Database\\SanDiego.gdb"

# Place all the rows from the MajorAttractions feature class
# into a Search cursor.
cur = arcpy.SearchCursor("MajorAttractions")

# Iterate through the cursor and print the name
# address, city, state and zip of each MajorAttraction
# to the Interactive Window.
# Print it in a standard three line address format.
for row in cur:
    print row.NAME
    print row.ADDR
    print row.CITYNM + ", CA " + row.ZIP
    # Alternate way
    # print row.NAME + "\n" + row.ADDR + "\n" + row.CITYNM + ", CA " + row.ZIP
```

UpdateField.py

```
# Author: ESRI
# Purpose: Add a field to a feature class and update that
#          field using the values from two other fields.

# Import the arcpy site package.
import arcpy

# Set the workspace.
arcpy.env.workspace = "C:\\Student\\PYTH\\Database\\Redlands.gdb"

# Add new field to Hospitals feature class.
arcpy.AddField_management("Hospitals", "Full_Address", "Text")
```

```
# Create update cursor for all rows in Hospital feature class.
cur = arcpy.UpdateCursor("Hospitals")

for row in cur:
    row.Full_Address = row.ARC_Street + ", " + row.CITY
    cur.updateRow(row)

# delete cursor variable to remove locks
del cur
```

CheckAndUpdateField.py

```
# Author: ESRI
# Purpose: Add a field to a feature class only if it does NOT already exist
#          and update that field using the values from two other fields.

import arcpy

# Set workspace.
arcpy.env.workspace = "C:\\Student\\PYTH\\Database\\Redlands.gdb"

#Variables for script
inputFC = "Hospitals"
field = "Full_Address"

# Check to see if the field already exists.
lstFlds = arcpy.ListFields(inputFC, field)

# Check for items in list object.
if len(lstFlds) == 0:
    # Add new field to Hospitals feature class.
    arcpy.AddField_management(inputFC, field, "Text")
else:
    print "Field already exists"

# Create update cursor for all rows in Hospital feature class.
cur = arcpy.UpdateCursor(inputFC)

for row in cur:
    row.setValue(field, row.ARC_Street + ", " + row.CITY)
    cur.updateRow(row)

# delete cursor variable to remove locks
del cur
```

7 Running your scripts in ArcToolbox

Introduction

Up to this point, you have mostly been writing scripts that use hard-coded paths and variables with hard-coded values. This can work fine for scripts that need to be run on the same set of data from time to time, but what if you want to run the script multiple times and specify a different set of input or output data? Or what if the data was to move to a different storage location?

By adding arguments for the paths and variables to your scripts, you can make the script more dynamic and better fit the scripts into your automation work flows. A script that has been made dynamic can be added to ArcToolbox as a custom script tool. This process allows you to define the arguments for the script, which are supplied by the script tool dialog box at run time.

Learning objectives

After completing this lesson, you will be able to:

- Write scripts that are dynamic
- Run scripts with arguments
- Attach a script to a custom tool

Key terms

- **Script tool**: A custom Python script that runs in ArcToolbox
- **Parameter**: An argument (both terms are used interchangeably)

What are some advantages to running your script within ArcGIS Desktop?

Notes ✏️

Making scripts dynamic

Scripts can be static or dynamic.

- A *static* script contains hard-coded values for paths and variables. The script is written to work with those values and does not allow for any change to those values at runtime.
- A *dynamic* script allows the user who is running the script to provide the values for the paths and variables at runtime. This makes the script more flexible to changes and easily adaptable to running scenarios and "what-ifs."

When scripts contain hardcoded values, such as feature class names or paths to data, the script knows exactly where to go to access the data and work with the defined values. But what if you want to run the script multiple times, specifying a different feature class or data location each time? Or what if you want the end user to select the feature class before running the script?

You can make your input and output values in your script dynamic by adding arguments to the script.

Note: You may notice that arguments are also referred to as parameters; both words are used interchangeably.

There are two functions that you can use to create an argument in your script.
Assign one of the following functions to any variable or value in your script:

▪ **`arcpy.GetParameterAsText()`**	The user's first argument always starts at 0. For example: `arcpy.GetParameterAsText(0)`
▪ **`sys.argv[]`**	The user's first argument starts at 1. For example: `sys.argv[1]`

You can execute scripts that use these functions via the methods indicated below.

Execute script via:	`sys.argv[]`	`arcpy.GetParameterAsText()`
Operating system prompt	✓	✓
PythonWin	✓	✓
Tool in ArcToolbox	✓	✓
Python window		✓
Server geoprocessing task		✓

There are additional advantages to using `arcpy.GetParameterAsText()` in your scripts.

- Your script can be run in Windows, Linux, or Unix.
- There is no limit to the number of characters that it will accept, making it very useful for long MultiValue inputs.

> **Note:** The sys.argv[] function has a limit of 1024 characters. If you are using a script tool that accepts MultiValue inputs, use the GetParameterAsText() function instead.

- You do not need to import the sys module, which results in less overhead.
- Your script will be more readable if you use the `SetParameterAsText` function in conjunction with the `GetParameterAsText` function.
  (Both of these ArcPy functions start their parameter numbering at 0.)

Running scripts with arguments

The ArcPy site package contains the `GetParameterAsText()` function to pass arguments to the script as parameters at runtime, which are assigned to variables in the script. By using GetParameterAsText() in your script, you can remove any hard-coded values or paths and replace them with arguments.

1. When you run a script tool in ArcToolbox, what benefits does the tool dialog box provide?

2. When arguments are passed to a script that is run in PythonWin, how do you indicate the separation of the arguments?

Attaching a script to a custom tool

You can run all your scripts from PythonWin or the operating system prompt; however, there are advantages to attaching a script to a tool in ArcToolbox.

1. **The script becomes part of the geoprocessing framework.**
   - You can run the script from a model, from the Python window, from a menu, or even from another script.
   - Your script tool will honor the current geoprocessing environment settings and the user will be able to get standard help documentation for that tool (i.e., scripting syntax, usage, etc.).
2. **The script receives a user-friendly interface.** For example:
   - If users want to enter an input feature class, they do not need to type the full path; they can use the Browse button.
   - If users want to build a SQL statement, they do not need to manually type the statement; they can use the Query Builder button.
3. **The tool's dialog box prevents many errors.** For example:
   - If the user enters the name of an input feature class that does not exist (or an output feature class that already exists), the dialog box informs the user.
   - If the parameter requires a specific type of data (e.g., a feature class), the dialog box will only let the user enter that type of data. The dialog box also restricts what the user can enter via values obtained from other parameters. Filters provide a valid set of values, and values obtained from other parameters can populate field lists and SQL dialog boxes with proper inputs.

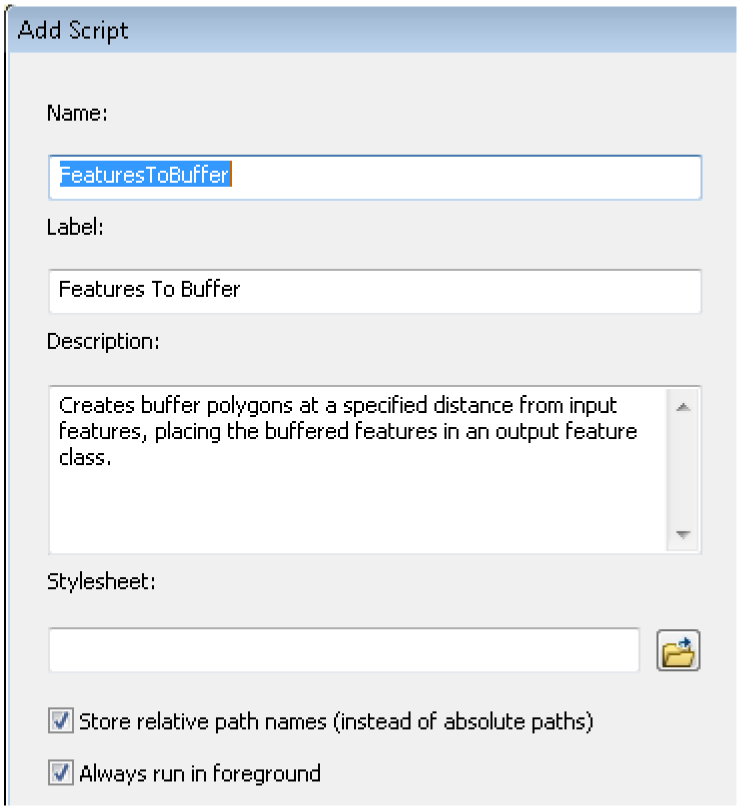
The Add Script wizard

Store all custom script tools in a custom toolbox—you cannot add a custom tool to a system toolbox. For detailed steps to add a script tool, in the ArcGIS Desktop Help, navigate to:

- Professional Library >
- Geoprocessing >
- Creating tools >
- Creating script tools with Python scripts >
- Adding a script tool

To create a script tool, right-click the custom toolbox and choose Add > Script.
The Add Script wizard contains three panels.

General properties
Provide the script tool's name, display label, description (for the help text), and stylesheet (the default stylesheets are sufficient for most purposes).


Add Script

Name: FeaturesToBuffer

Label: Features To Buffer

Description: Creates buffer polygons at a specified distance from input features, placing the buffered features in an output feature class.

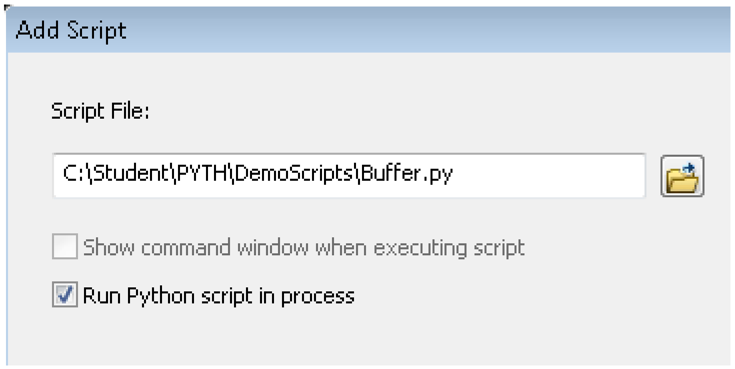
Stylesheet:

[✓] Store relative path names (instead of absolute paths)

[✓] Always run in foreground

Script location
Provide the file that will be executed from the tool (i.e., Python script, AML, or EXE).

You can also choose to show the command window and run a Python script in-process.


Add Script

Script File: C:\Student\PYTH\DemoScripts\Buffer.py

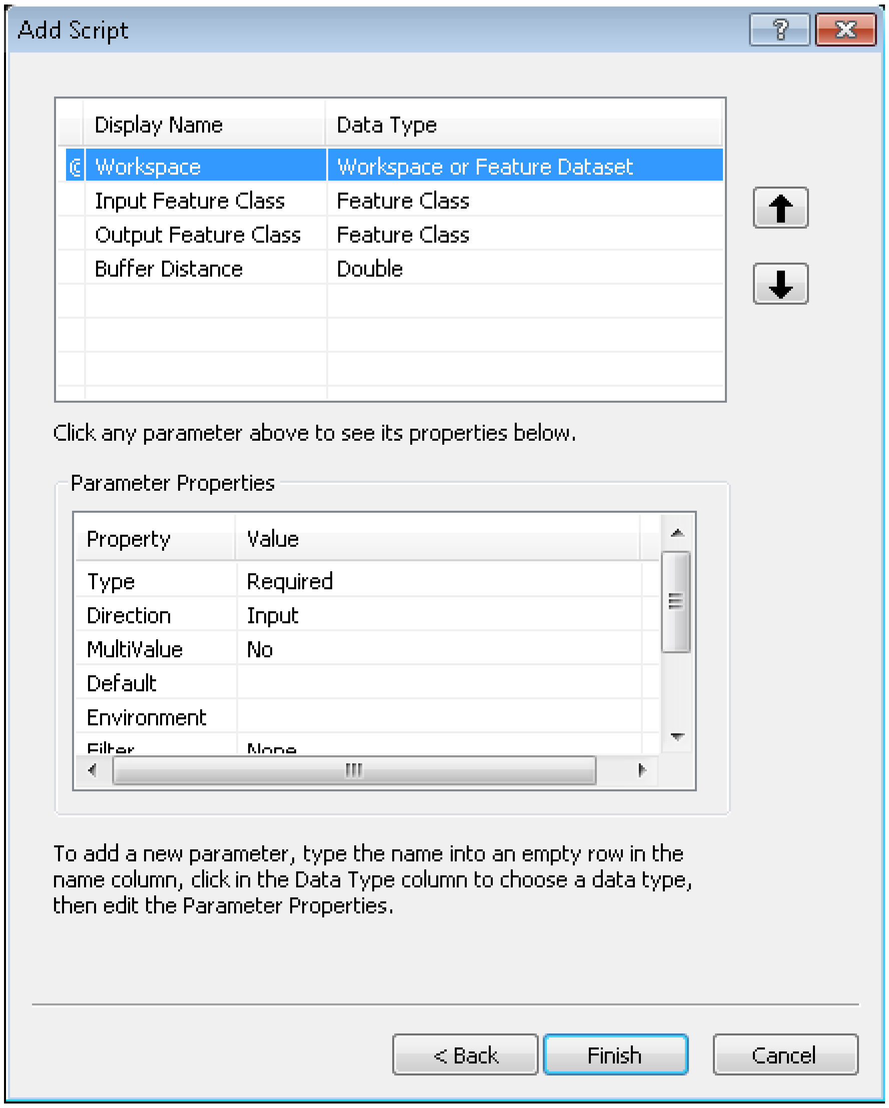
[ ] Show command window when executing script

[✓] Run Python script in process

Parameter properties
Provide the parameters that correspond to each `sys.argv[ ]` or `GetParameterAsText()` argument in the script.

Enter parameters in the same order as in the script.

Each parameter has additional properties that you can set.



Display Name	Data Type
Workspace	Workspace or Feature Dataset
Input Feature Class	Feature Class
Output Feature Class	Feature Class
Buffer Distance	Double

Click any parameter above to see its properties below.

Parameter Properties

Property	Value
Type	Required
Direction	Input
MultiValue	No
Default	
Environment	
Filter	None

To add a new parameter, type the name into an empty row in the name column, click in the Data Type column to choose a data type, then edit the Parameter Properties.

For more information about parameter properties, in the ArcGIS Desktop Help, navigate to:

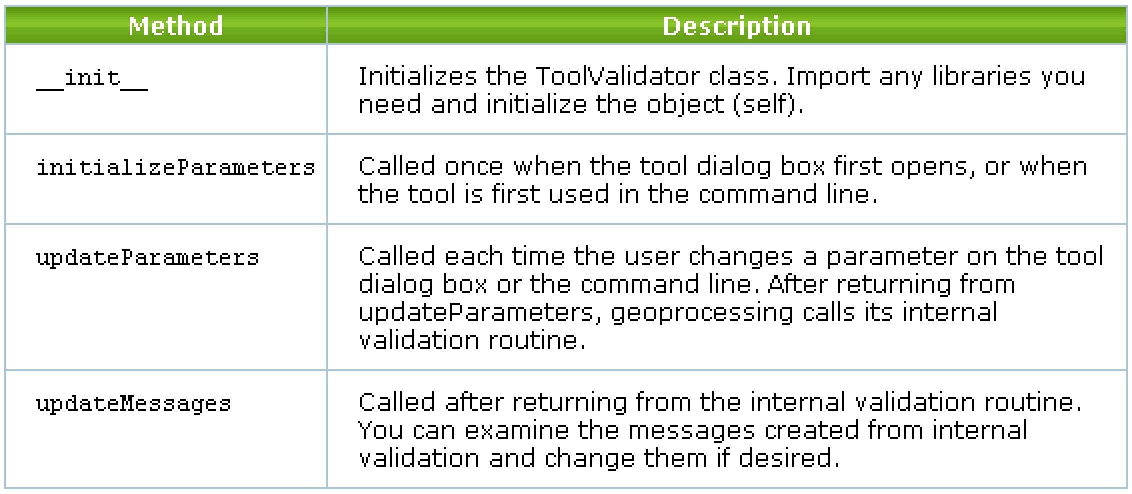
- Professional Library >
- Geoprocessing >
- Creating tools >
- Creating script tools with Python scripts >
- Setting script tool parameters

ToolValidator

All script tools come with the ToolValidator class, which enables the validation of parameters. This could include setting default values for parameters based on what you set for other parameters or the license available, or enabling or disabling a parameter based on the tool dialog inputs.

If the script tool is placed in a model, the ToolValidator can create the required schema for describing the derived output of the tool to the model and give you the same capabilities of a system tool in a model, such as providing a derived output from the tool.

The following table from the ArcGIS Desktop Help shows the ToolValidator methods found on all script tools:



Method	Description
`__init__`	Initializes the ToolValidator class. Import any libraries you need and initialize the object (self).
`initializeParameters`	Called once when the tool dialog box first opens, or when the tool is first used in the command line.
`updateParameters`	Called each time the user changes a parameter on the tool dialog box or the command line. After returning from updateParameters, geoprocessing calls its internal validation routine.
`updateMessages`	Called after returning from the internal validation routine. You can examine the messages created from internal validation and change them if desired.

ToolValidator methods overview

While you cannot edit the ToolValidator for a system script tool, you can edit the ToolValidator class methods for any script tool located in a custom toolbox.

Code samples

For your reference and study, here is some sample code that shows the use of the updateParameter method and the updateMessages method on the ToolValidator class, taken from the Hot Spot tool and the ArcGIS Desktop Help.

Enable or disable a parameter

```
def updateParameters(self):

  # If the option to use a weights file is selected (the user chose
  #  "Get Spatial Weights From File", enable the parameter for specifying
  #  the file, otherwise disable it
  #
  if self.params[3].value == "Get Spatial Weights From File":
    self.params[8].enabled = 1
  else:
    self.params[8].enabled = 0
```

Set a default value

```
def updateParameters(self):
  # Set the default distance threshold to 1/100 of the larger
  # of the width or height of the extent of the input features.
  # Do not set if there is no input dataset yet, or the user
  # has set a specific distance (Altered is true).
  import string

  if self.params[0].value:
    if not self.params[6].altered:
      extent = string.split(arcpy.Describe(self.params[0].value).extent, " ")
      width = float(extent[2]) - float(extent[0])
      height = float(extent[3]) - float(extent[1])
      if width > height:
        self.params[6].value = width / 100
      else:
        self.params[6].value = height / 100
   return

def updateMessages(self):
    return
```

Customize a message

```
def updateMessages(self):
    self.params[6].clearMessage()

  # Check to see if the threshold distance contains a value of zero
  # and the user has specified a fixed distance band.
  #
  if self.params[6].value <= 0:
    if self.params[3].value == "Fixed Distance Band":
      self.params[6].setErrorMessage("Zero or a negative distance is invalid \
                                      when using a fixed distance band. Please \
                                      use a positive value greater than zero." )
      elif self.params[6].value < 0:
        self.params[6].setErrorMessage("A positive distance value is required \
                                        when using a fixed distance band. \
                                        Please specify a distance.")
   return
```

Update a parameter's filter property

This code example dynamically updates a Value List Filter containing a choice list of keywords.

- If the user enters "OLD_FORMAT" in the second parameter, the third parameter contains "POINT, LINE, and POLYGON."
- If the user enters "NEW_FORMAT," the third parameter contains three additional choices.

```
class ToolValidator:
  def __init__(self):
    import arcpy
    self.params = arcpy.GetParameterInfo()

  def initializeParameters(self):
    return

  def updateParameters(self):
    # Provide default values for "file format type" and
    #  "feature type in file"
    #
    if not self.params[1].altered:
      self.params[1].value = "OLD_FORMAT"
    if not self.params[2].altered:
      self.params[2].value = "POINT"
    # Update the value list filter of the "feature type in file" parameter
    #   depending on the type of file (old vs. new format) input
    #
    if self.params[1].value == "OLD_FORMAT":
      self.params[2].filter.list = ["POINT", "LINE", "POLYGON"]
    elif self.params[1].value == "NEW_FORMAT":
      self.params[2].filter.list = ["POINT", "LINE", "POLYGON",
                                    "POINT_WITH_ANNO",
                                    "LINE_WITH_ANNO",
                                    "POLYGON_WITH_ANNO"]

    return

  def updateMessages(self):
    return
```

Exercise 7A: Create a script tool to copy features

Estimated time: 45 minutes

In this exercise, you will make dynamic input and output arguments for variable values. You will also learn how to run scripts with dynamic arguments from PythonWin, the Python window in ArcMap, and in ArcToolbox.

In this exercise, you will:

- Run scripts using hard-coded and dynamic values
- Run a script from PythonWin
- Attach a script to a tool
- Run script tools from ArcToolbox and in the Python window

Step 1: Run a script using hard-coded values

In this step, you will run a script using hard-coded values. In the next step, you will replace the hard-coded values with dynamic arguments.

❑ If necessary, open PythonWin.

❑ Open the C:\Student\PYTH\Exercise07\FC2FC_HardCoded.py script.

This script copies selected features from one feature class into a new feature class. The `MakeFeatureLayer` tool makes a temporary layer that holds selected features from the C:\Student\PYTH\Database\World\Country.shp. The `CopyFeatures` tool copies the selected features into the new Country feature class.

❑ Read through the script's comments. Make sure you understand the script before you continue. If you are having difficulties, ask your instructor for help.

Notice that the following input and output values are hard-coded in this script—The user has no input or output choices:

- `inFC`
- `outFC`
- `expression`
- `fieldInfo`

❑ Run the script.

- ❑ Open ArcCatalog and verify that the C:\Student\PYTH\Database\World.gdb\Country feature class was created.

This script copied all the features from the ...\Database\World\Country.shp into the ...\Database\World.gdb\Country feature class. The script did the job that it was supposed to do. However, this script has limited use because all the values are hard-coded. This script would suit a wider audience if all input and output arguments were dynamic.

In the next step, you will make this script more flexible.

- ❑ Minimize ArcCatalog.

Step 2: Replace hard-coded values with dynamic values

In this step, you will replace the hard-coded values with the `arcpy.GetParameterAsText()` function to make the variable values dynamic. This makes the script more flexible and appealing to other users.

- ❑ Save the FC2FC_HardCoded.py script as **FC2FC.py**.

- ❑ Modify the following variables to accept user-specified input:

```
inFC = arcpy.GetParameterAsText(0)
outFC = arcpy.GetParameterAsText(1)
expression = arcpy.GetParameterAsText(2)
fieldInfo = arcpy.GetParameterAsText(3)
```

- ❑ Save the script.

Now you can run the script with dynamic inputs.

Step 3: Run a script with arguments from PythonWin

In this step, you will run the FC2FC.py script from PythonWin. When running the script, you must enter all four expected arguments. The arguments need to be passed into the script in the order in which the variable values are set, separated by spaces. If you do not want to pass a value for an argument, simply type an empty string.

- ❑ On the Standard toolbar, click the Run button .

You will copy all the features in the ...\Database\World\Cities.shp into a new feature class in the ...\Database\World.gdb.

- ❑ In the Run Script dialog box, for Arguments, type the following arguments on a *single* line, separated by spaces:

```
C:\Student\PYTH\Database\World\Cities.shp
C:\Student\PYTH\Database\World.gdb\Cities
""
""
```

- ❑ Click OK to run the script.
- ❑ When the script has finished executing, verify that the C:\Student\PYTH\Database\World.gdb\Cities feature class was created.

This script ran and performed the job that it was supposed to do. However, this time the script accepted dynamic input from the user, making the script more flexible. In the remaining steps, you will learn how to run a script with arguments from other environments, including ArcToolbox and the Python window.

Step 4: Attach a script with arguments to a tool in ArcToolbox

In this step, you will run a script with arguments from a tool in ArcToolbox. Once a script is part of ArcToolbox, it behaves like any other tool. This means you can run the script as a dialog box, from the Python window, or from a model. You can also attach help documentation to the script and run the script using the Geoprocessing environment settings.

- ❑ If necessary, open ArcCatalog.
- ❑ Browse to C:\Student\PYTH\Exercise07 folder.

Before you add a script to ArcToolbox, you need to create a toolbox to store the script.

- ❑ In the Catalog tree, right-click and choose New Toolbox.
- ❑ Rename the toolbox **Custom Tools.tbx**.
- ❑ Right-click Custom Tools and choose Properties.
- ❑ In the Custom Tools Properties dialog box, for Alias, type **custom**.
- ❑ Click OK.

Using an alias can help to avoid confusion between tool names.

Next, you will attach the FC2FC.py script to a tool in the Custom Tools toolbox.

❑ Right-click the Custom Tools toolbox and choose Add > Script.

The first panel of the Add Script wizard asks for the script tool's Name, Label, and Description. The Name is used to execute the script tool from the Command Line window or from another script. The Name cannot contain spaces. The Label is the display name for the script (i.e., how it will appear in the ArcToolbox window). Labels may contain spaces.

❑ In the Add Script wizard, set the following:
- Name: **FeaturesToFeatures**
- Label: **Feature class to feature class**
- Description: **Copy selected features from one feature class to a new feature class. Fields from the input feature class can be altered in the output feature class.**
- Store relative path names: Check the box.
- Always run in foreground: Uncheck the box.

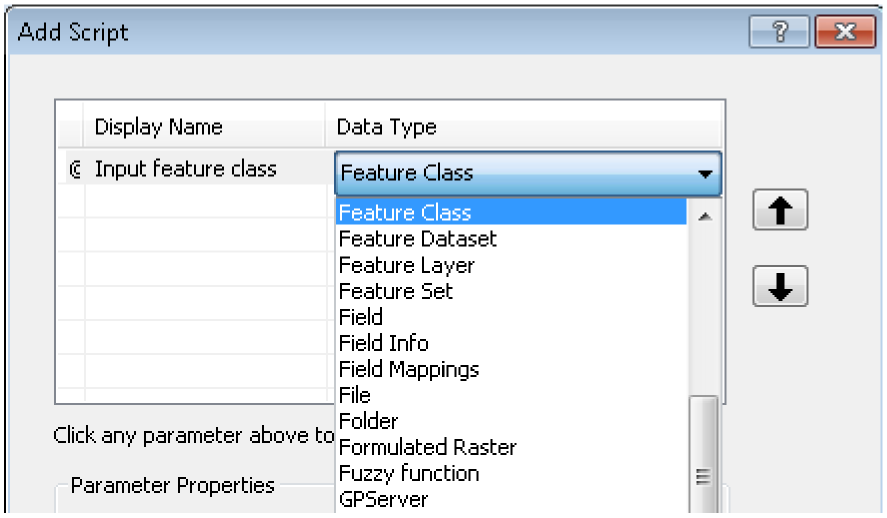
❑ Click Next.

❑ For Script File, browse to and open the C:\Student\PYTH\Exercise07\FC2FC.py script.

❑ Click Next.

Next, you will enter the four script arguments and their parameter properties.

❑ For the first argument, set the following:
- Display Name: **Input feature class**
- Data Type: Feature Class



- ❑ For the second argument, set the following:
  - Display Name: **Output feature class**
  - Data Type: Feature Class
  - Parameter Properties:
    - Direction: Output

The output feature class does not exist (it will be created during script execution); therefore, the Direction must be defined as Output.

- ❑ For the third argument, set the following:
  - Display Name: **Expression**
  - Data Type: SQL Expression
  - Parameter Properties:
    - Type: Optional
    - Obtained From: Input_feature_class

Changing the Parameter Type to Optional means that the user does not need to specify a SQL expression. In this case, all features in the input feature class will be copied to the output feature class. If the user enters a SQL expression, only the selected features will be copied to the output feature class.

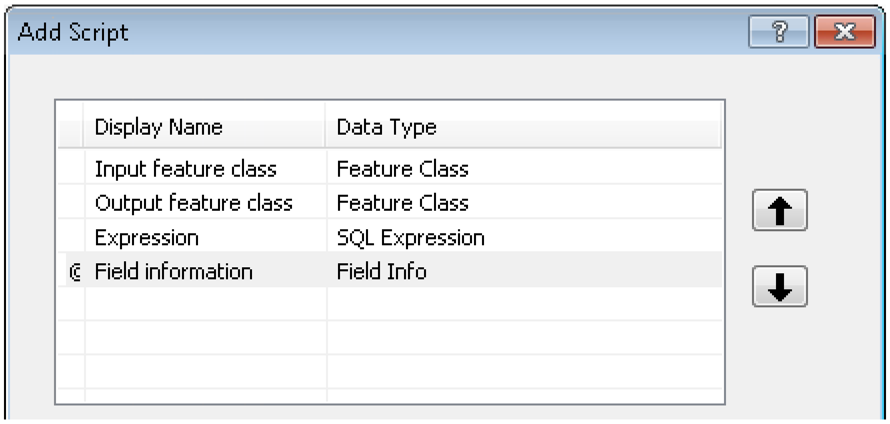
The Expression will obtain the attributes from the Input feature class to populate the Query Builder dialog box. If you leave the Obtained From field blank, the user would need to type the expression.

- ❑ For the fourth argument, set the following:
  - Display Name: **Field information**
  - Data Type: Field Info
  - Parameter Properties:
    - Type: Optional
    - Obtained From: Input_feature_class

Changing the Parameter Type to Optional means that the user does not need to specify which fields to alter. In this case, all fields in the input feature class will be unaltered in the output feature class.

The Field information will obtain a list of fields from the Input feature class and populate the field list.

- ❑ Verify that your list of script arguments matches the following:



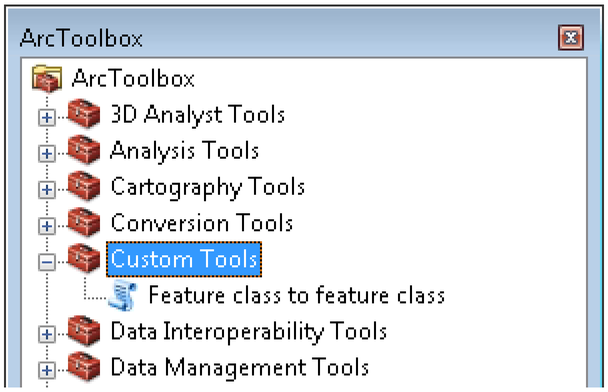
Display Name	Data Type
Input feature class	Feature Class
Output feature class	Feature Class
Expression	SQL Expression
Field information	Field Info

- ❑ Click Finish.

The script tool is now ready for use.

- ❑ Add the new custom toolbox to ArcToolbox:
  - If necessary, open the ArcToolbox window.
  - Right-click the top level ArcToolbox and select Add Toolbox.
  - Browse to your custom toolbox in the Exercise07 folder, select the toolbox and click Open.

- ❑ Expand the Custom Tools toolbox.



In the next step, you will run the script tool from ArcToolbox.

Step 5: Run a script tool from ArcToolbox

In this step, you will run your Feature class to feature class script tool.

- ❑ In ArcToolbox, double-click the Feature class to feature class script tool to open it.

The dialog box prompts you for the four arguments that you specified in the Add Script wizard. Two arguments are optional: Expression and Field information.

- ❑ In the Feature class to feature class dialog box, click Show Help.

The description that you wrote for the tool displays in the tool's help panel.

- ❑ Click Hide Help to collapse the help panel.

- ❑ For Input feature class, click the Browse button.

The Browse button is available because you assigned this argument the Feature Class data type. The tool's dialog box is smart enough to supply a Browse button for feature classes, feature datasets, workspaces, and so on.

- ❑ In the Input feature class dialog box, browse to and add the C:\Student\PYTH\Database\World\Country.shp.

Notice that the field names are added to the Field information list. This is because you set the Obtained From parameter property to the Input feature class and also the Field information parameters.

- ❑ For Output feature class:
  - Browse to the C:\Student\PYTH\Database\World.gdb\Mongolia feature dataset.
  - Name the new feature class **Mongolia**.

- ❑ For Expression, click the SQL button.

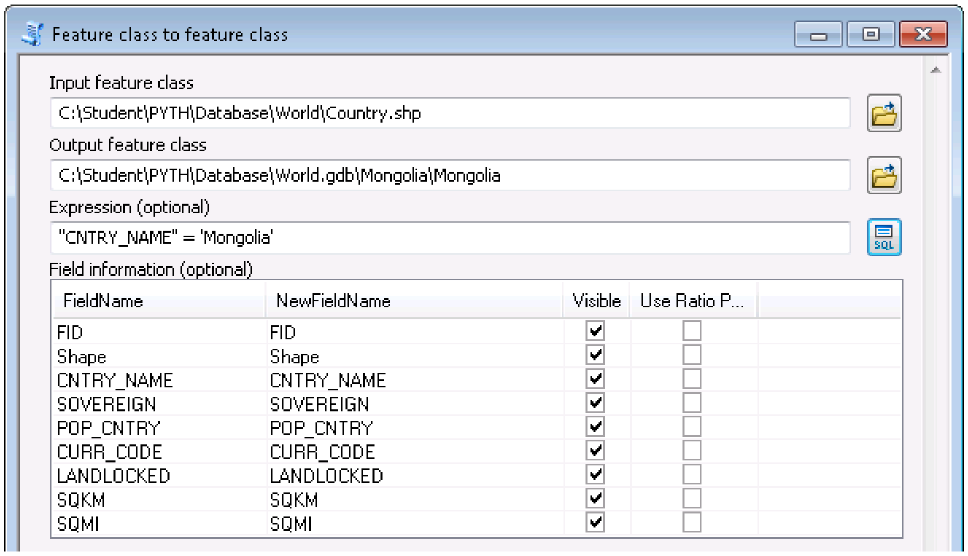
- ❑ In the Query Builder dialog box, create the following SQL statement.

```
"CNTRY_NAME" = 'Mongolia'
```

Notice that all the field and attribute values are available in the Query Builder dialog box. This is because you set the Obtained From parameter property between the Input feature class and the Expression.

- ❑ Click OK.

❑ Verify that your Feature class to feature class dialog box matches the following graphic:



Input feature class
C:\Student\PYTH\Database\World\Country.shp
Output feature class
C:\Student\PYTH\Database\World.gdb\Mongolia\Mongolia
Expression (optional)
"CNTRY_NAME" = 'Mongolia'
Field information (optional)

FieldName	NewFieldName	Visible	Use Ratio P...
FID	FID	✓	☐
Shape	Shape	✓	☐
CNTRY_NAME	CNTRY_NAME	✓	☐
SOVEREIGN	SOVEREIGN	✓	☐
POP_CNTRY	POP_CNTRY	✓	☐
CURR_CODE	CURR_CODE	✓	☐
LANDLOCKED	LANDLOCKED	✓	☐
SQKM	SQKM	✓	☐
SQMI	SQMI	✓	☐

❑ Click OK.

This tool will copy the Mongolia feature in the ...\Database\World\Country.shp to a new feature class in the ...\Database\World.gdb\Mongolia feature dataset.

❑ Verify that the Mongolia feature class was created.

Your script tool is now part of the geoprocessing framework. At this point, you can run the tool from the Python window, add the tool to a model, run the script from another script, or assign help documentation to the tool.

Step 6: Run a script tool from the Python window

In this step, you will run your script tool in the Python window.

Before running your script tool, you must first tell the Python window in which toolbox your custom tool is located. The Python window is aware of all system toolboxes and the tools that they contain, but is not aware of custom tools. You will import your toolbox into the Python window and then run the script tool.

❑ Open the Python window (Geoprocessing > Python). Move the Python window to a location where you can see the entire window.

By default, the Python window imports the ArcPy site package, so you are ready to issue Python commands, statements, and ArcPy commands.

- ❑ In the Python window, at the >>> prompt, type the following code:

```
arcpy.ImportToolbox(r"C:\Student\PYTH\Exercise07\Custom Tools.tbx")

arcpy.FeaturesToFeatures_custom("C:\Student\PYTH\Database\World\Lakes.shp",
                                 "C:\Student\PYTH\Database\World.gdb\Lakes")
```

- ❑ Verify that the Lakes feature class has been created in the World.gdb file geodatabase.
- ❑ Close the Python window.

Exercise 7B: Buffer multiple feature classes

Estimated time: 30 minutes

In the previous exercise, you learned how to attach a script to a script tool. In this exercise, you will attach another script to a script tool, and you will use the `arcpy.GetParameterAsText()` function. You will also learn how to set the Filter, MultiValue, and Default properties for an argument in a script tool.

In this exercise, you will:

- Replace hard-coded values with dynamic values in a script
- Attach a script to a tool in ArcToolbox
- Run a script tool

Step 1: Replace hard-coded values with dynamic values

In this step, you will replace hard-coded values with dynamic arguments.

- ❑ In PythonWin, open the C:\Student\PYTH\Exercise07\Buffer_HardCoded.py file.

- ❑ Save the Buffer_HardCoded.py script as **Buffer.py**.

This script buffers multiple feature classes at a specified distance. All the values are currently hard-coded. You will make these values dynamic using the `arcpy.GetParameterAsText()` function.

- ❑ Modify the following variables to accept dynamic arguments for values.
  - `inFCs`
  - `outWS`
  - `dist`

- ❑ Save the script.

Step 2: Attach a script with arguments to a tool in ArcToolbox

In this step, you will attach the Buffer.py script to a tool in the Custom Tools toolbox. In the previous exercise, you used the Type, Direction, and Obtained From properties. In this step, you will use the Type, Direction, MultiValue, Default, and Filter properties.

- ❑ Return to ArcCatalog.

- ❑ Right-click within ArcToolbox and choose Environments.
- ❑ Expand Workspace and set the Current Workspace to **C:\Student\PYTH**.
- ❑ Click OK to close the Environment Settings dialog box.
- ❑ In ArcToolbox, right-click the Custom Tools toolbox and choose Add > Script.
- ❑ In the Add Script wizard, set the following:
  - Name: **BufferMultipleFC**
  - Label: **Buffer multiple feature classes**
  - Description: **Creates buffers for multiple feature classes. Buffer distance must be between 500 - 1500 feet.**
  - Store relative path names: Check the box.
  - Always run in foreground: Uncheck the box.
- ❑ Click Next.
- ❑ For Script File, browse to and add the C:\Student\PYTH\Exercise07\Buffer.py script.
- ❑ Verify that Run Python script in process is checked.

Running the script tool in process will ensure that the Python script does not start a separate process to run. This can enhance the performance of your script tools when run in ArcGIS Desktop.

- ❑ Click Next.

Now you will specify the script arguments and their properties. Three variables need to be set as arguments: `inFCs`, `outWS`, and `dist`.

- ❑ For the first argument, set the following:
  - Display Name: **Feature classes to buffer**
  - Data Type: Feature Class
  - Parameter Properties:
    - MultiValue: Yes

Setting the MultiValue property to Yes allows the script to buffer multiple feature classes instead of just one.

❑ For the second argument, set the following:
- Display Name: **Output location**
- Data Type: Workspace or Feature Dataset
- Parameter Properties:
  - Environment: Current Workspace [workspace]

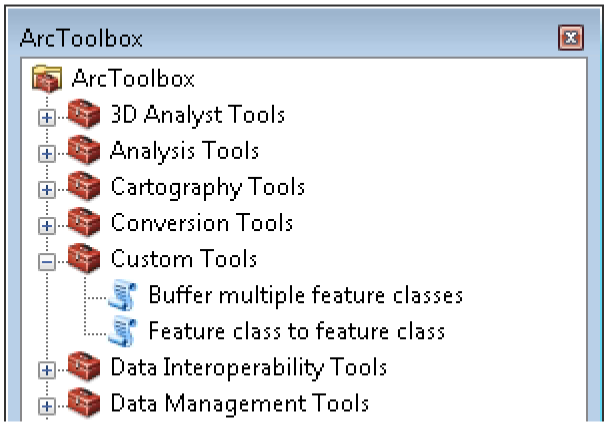
If the user does not specify an output location for the BufferMultipleFC tool, the output will be sent to the C:\Student\PYTH folder.

❑ For the third argument, set the following:
- Display Name: **Distance**
- Data Type: Double
- Parameter Properties:
  - Filter: Range

❑ In the Range dialog box that opens, set the following then click OK:
- Minimum Value: **500**
- Maximum Value: **1500**

By setting a range filter, you are restricting the user to entering a buffer distance between 500 and 1500 feet.

❑ In the Add Script wizard, click Finish.



Your new script tool is now ready for use.

Step 3: Run the BufferMultipleFC script tool

In this step, you will test your new script tool.

- ❑ In ArcToolbox, double-click the Buffer multiple feature classes script tool.
- ❑ In the Buffer multiple feature classes dialog box, for Feature classes to buffer, browse to the C:\Student\PYTH\Database\SanDiego.gdb and add the Railroads and Freeways feature classes.

Notice that you can add multiple feature classes to the list. This is because you set the MultiValue property to Yes for the Feature classes to buffer. Also notice that the Output location displays a default value. This is another property that you set when you created the script tool.

- ❑ For Output location, accept the default value.
- ❑ For Distance, type **2000**.
- ❑ Click OK.

An error displays and a red X appears next to the Distance label.

1. Why did the script tool fail to run?

____

- ❑ Click OK to dismiss the error message.
- ❑ Enter a valid Distance value and click OK.
- ❑ If necessary, close the progress window when the tool has finished executing.
- ❑ Verify that the two new feature classes were created in the C:\Student\PYTH folder.
- ❑ Preview the two new feature classes, then delete them.
- ❑ Close ArcCatalog.

Lesson review

1. How do you make a Python script dynamic?

2. The ToolValidator code can be edited in a system script tool.
   a. True
   b. False

Answers to Lesson 7 questions

Running scripts with arguments

1. When you run a script tool in ArcToolbox, what benefits does the tool dialog box provide?

   The tool dialog box validates all inputs, checks to make sure the output data does not exist, offers default values, and has a much shorter startup time.

2. When arguments are passed to a script that is run in PythonWin, how do you indicate the separation of the arguments?

   You separate the arguments with spaces.

Exercise 7B: Buffer multiple feature classes

1. Why did the script tool fail to run?

   The distance value is out of the range from 500 to 1500.

Lesson review

1. How do you make a Python script dynamic?

   You replace hard-coded values and paths with arguments.
   `arcpy.GetParameterAsText()` can pass arguments to your script.

2. The ToolValidator code can be edited in a system script tool.

   b. False

Exercise solution: 7A

FC2FC.py

```
# Author: ESRI
# Date:
# Purpose: Copy selected features from one feature class into
#          a brand new feature class.

# Import the arcpy and os.path modules.
import arcpy
import os

# Input feature class.
inFC = arcpy.GetParameterAsText(0)
# Output feature class.
outFC = arcpy.GetParameterAsText(1)
# SQL statement to filter features. If the expression is empty,
# all features will be copied into the new feature class.
expression = arcpy.GetParameterAsText(2)
# Field information to alter output fields. If the fieldInfo
# is empty, all fields will be unaltered in the new feature class.
fieldInfo = arcpy.GetParameterAsText(3)

# Make a temporary layer to hold the selected features and altered fields.
# The selected features are determined by the expression variable.
# The altered fields are determined by the fieldInfo variable.
arcpy.MakeFeatureLayer_management(inFC, os.path.basename(outFC), expression,
                               os.path.dirname(outFC), fieldInfo)

# Copy selected features and altered fields to the output feature class.
arcpy.CopyFeatures_management(os.path.basename(outFC), outFC)

# Remove any database locks by deleting objects
del arcpy, inFC, outFC
```

Exercise solution: 7B

Buffer.py

```
# Author: ESRI
# Date:
# Purpose: Creates buffers around specified feature classes.
#          Buffer distance is restricted to 500 - 1500 feet.

# Import the arcpy and os modules.
import arcpy
import os

# Input feature classes to buffer.
inFCs = arcpy.GetParameterAsText(0)

# Output workspace.
outWS = arcpy.GetParameterAsText(1)

# Buffer distance.
dist = arcpy.GetParameterAsText(2)

# Split input feature classes into separate feature classes.
inFCs = inFCs.split(";")

# Loop through each feature class and create buffers.
for inFC in inFCs:
    # Figure out the name of the output feature class.
    (filePath, fileName) = os.path.split(inFC)
    dotInd = fileName.find(".")
    if dotInd <> -1:
        newFC = fileName[0:dotInd]
        outFC = newFC + "_buffer"
    else:
        outFC = fileName + "_buffer"

    # Create the buffer feature class
    arcpy.Buffer_analysis(inFC, outWS + "\\" + outFC, str(dist) + " Feet")
```

8 Handling Python and ArcPy exceptions

Introduction

In Lesson 3, you worked with several different script debugging techniques. The techniques focused on Python syntax and logic errors, but did not explore errors or exceptions that could be generated by Python or by running the script in the geoprocessing framework.

In this lesson, you will examine how to handle an exception raised by Python, handle an arcpy ExecuteError exception, and work with the Python traceback module to obtain detailed information about the exception. Gracefully trapping for errors when the code executes makes your script look more professional and efficient and can reduce user frustration.

Learning objectives

After completing this lesson, you will be able to:

- Trap runtime script errors
- Handle Python exceptions
- Handle ArcPy exceptions
- Use Python traceback module

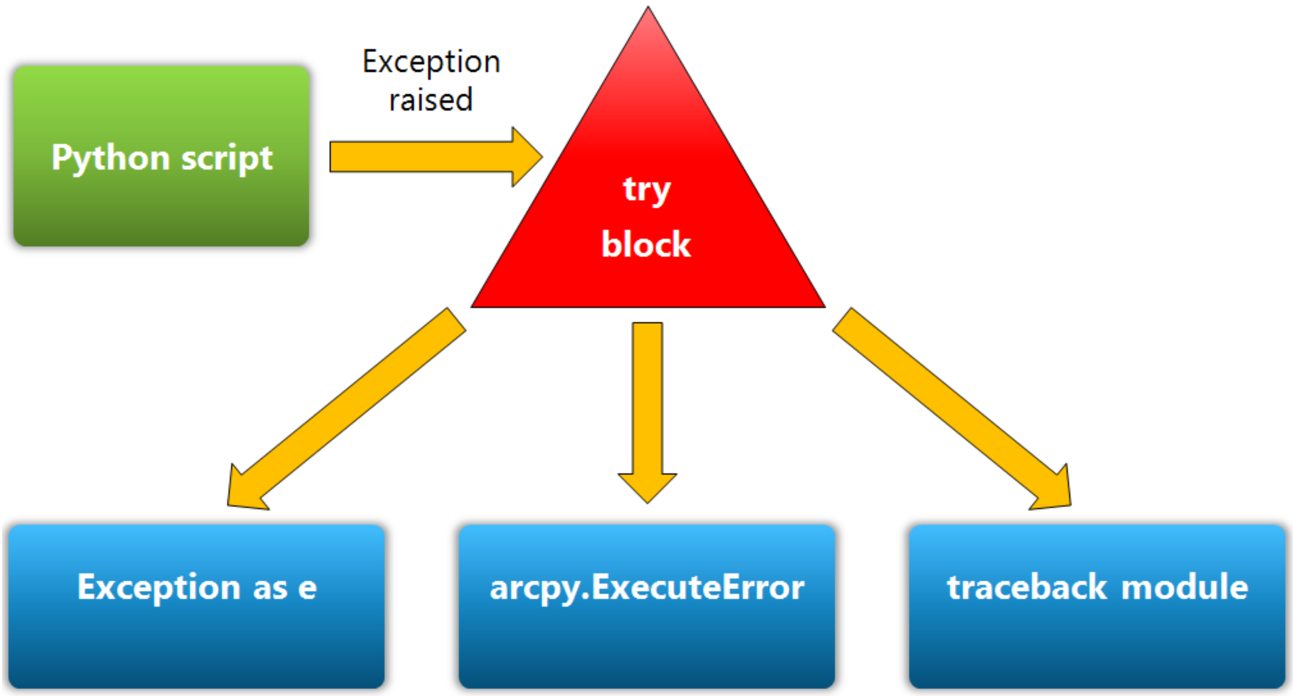
Key terms

- **Errors**:There are two types, syntax errors and exceptions. Syntax errors can be detected and corrected before the script is executed.
- **Exception:**: An error that is detected during execution of the script.

Handling exceptions in scripts

When writing scripts, you have found that it is almost impossible to avoid errors. You've learned that finding and fixing syntax errors is pretty straight forward, but finding errors that occur when the script executes can be a bit more challenging.

To help us out, Python differentiates between types of error. Errors can be broken down into two basic types: the syntax error and the exception. You worked with Python syntax errors in Lesson 3. In this lesson, you will work with exceptions. Exceptions are generated only when code is executed. An exception is simply an error that occurred as a result of executing code.



You can handle exceptions using the following techniques:

standard **except** block	catches any errors
Exception as e block	prints only the error message
arcpy.ExecuteError	catches only arcpy geoprocessing errors
traceback module in a standard **except** block	provides detailed Python messages and shows geoprocessing messages

Note: To learn more about error handling in scripts, refer to the ArcGIS Desktop Help:

- Professional Library >
- Geoprocessing >
- Geoprocessing with Python >
- Accessing tools >
- Error handling with Python

Using try..except

One of the simplest and possibly the easiest ways to catch exceptions in your scripts is to use the `try..except` block. The code that you want to run in your script is placed inside the try block, and code to handle the exception is placed in the except bock. When you execute (run) your script, if any exception is detected in your code within the try block, Python jumps down to the except block and executes that code.

The major advantage to using the `try..except` block is that your code will not fail with a traceback message if an exception is detected while executing the code. The except block handles the exception by running the code you have there.

Code that you place in the `except` block could include print statements for detailing the exception and, if the script is using the ArcPy site package, for printing geoprocessing messages.

Using Exception as e

Using the Python `Exception as e` statement can be useful when you need some details on the exception that occurred.

If the exception was generated by a Python error, you can print details of the exception by using `print e` in the exception block.

Here's an example of a simple `except` block using `Exception as e`:

```
try:
    #
    # Your code goes here
except Exception as e:
    print "Error occurred"
    print e
```

Using arcpy.ExecuteError

There may be times when you would like to handle an exception generated from the geoprocessor separately from any Python errors. The `arcpy.ExecuteError` and `arcpy.ExecuteWarning` exception classes can be raised when a geoprocessing tool encounters an error or warning.

When the execution of a geoprocessing tool encounters an error, the `arcpy.ExecuteError` exception class is raised. You can use the exception class in your script along with a print statement to display the error.

Here's an example of a simple `arcpy.ExecuteError` exception handler in a script:

```
try:
    #
    # Your code goes here
except arcpy.ExecuteError:
    print "Geoprocessing tool error occurred"
    print arcpy.GetMessages(2)
```

The `arcpy.GetMessages()` function returns messages generated by the geoprocessor while executing geoprocessing functions in your script. There are three levels of messages.

arcpy.GetMessages()

Severity Level	Messages returned
0	all messages
1	warning messages
2	error messages

Not specifying the severity level will return all messages.

Using the traceback module

The Python `traceback` module can be extremely useful when attempting to track down why your script failed and the line of code where the script failed. When `traceback` is used in conjunction with the `sys` module, you can isolate the exact location and cause of the exception. The traceback is handled in a standard `except` bock.

Here's an example of using the `traceback` module with the `sys` module:

```
try:
    # Your code goes here
except:
    # Get the traceback object
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]

    # Concatenate information together concerning the error
    # into a message string
    #
    pymsg = "PYTHON ERRORS:\nTraceback info:\n" + tbinfo + \
            "\nError Info:\n" + str(sys.exc_info()[1])
    msgs = "ArcPy ERRORS:\n" + arcpy.GetMessages(2) + "\n"

    # Return python error messages for use in script tool or
    # Python window
    #
    arcpy.AddError(pymsg)
    arcpy.AddError(msgs)

    # Print Python error messages for use in Python or
    # Python window
    #
    print pymsg + "\n"
    print msgs
```

In the code sample, when an exception occurs, the `except` block will get the traceback object and print out the traceback information, and then print any ArcPy error messages that had occurred. For compatability with scripts run in ArcGIS Desktop or as a server task, the `arcpy.AddError` function is used to pass the traceback and geoprocessing messages to the dialog box and results window.

Exercise 8: Working with exceptions

Estimated time: 45 minutes

Implementing error handling in your scripts is always a good idea. If your script encounters an exception, you can gracefully handle the error and not have the script crash on the end-user side. Adding code to your script to handle any runtime error is a good programming practice.

In this exercise, you will:

- Incorporate a try..except block into an existing script
- Use the Exception as e handler
- Use the arcpy.ExecuteError exception class
- Use the traceback and sys modules

Step 1: Incorporate try..except

All the exception handlers that you will work with in this exercise use a `try..except` block. The `try..except` block is a simple Python structure, in which the code that you want to run is placed within the `try` block and the code to handle the exception is placed within the `except` block.

- ❑ In PythonWin, open the C:\Student\PYTH\Exercise08\Ex08.py script and save it as **MyExceptionHandler.py**.
- ❑ Read the script comments so that you understand the script.
- ❑ Click the Check button to check the syntax, then run the script—it will fail with a traceback exception.
- ❑ In the Interactive Window, review the traceback messages.

The script failed with an exception and printed a lot of error messages. A good script always has code to handle exceptions and print more meaningful messages, which provide a more positive experience to the end user.

To print more meaningful messages and to prevent the script from crashing, you will add error handling code to the script.

- ❑ Using your knowledge and skills, incorporate a `try..except` block into the script.
- ❑ Within the `except` block, write code to add two print statements:

1. Inform the user that an error has occurred.
2. Return all geoprocessing messages (using the **`arcpy.GetMessages()`** function).

- ❑ Check your script syntax and fix any errors that are detected.
- ❑ Run your script.
- ❑ Review the messages that printed to the Interactive Window.

1. Why did the tool fail to execute?

____

2. What geoprocessing tool error has occurred?

____

3. Based on the error messages, how might you fix the problem?

____

____

____

In the next step, you will work with the `Exception as e` handler.

Step 2: Use Exception as e

You can use the `Exception as e` handler to print the exception messages to the Interactive Window. In addition to any Python exceptions, any geoprocessing errors will be handed by the exception handler. The exception messages can be printed to the Interactive Window or added to the Geoprocessing dialog window.

- ❑ Comment out the entire `except` block.
- ❑ Add a new **`except`** block using the **`Exception as e`** handler.

- [ ] Within the new `except` block, write code to:
  - Print a message to the Interactive Window to indicate that an error occurred
  - Print **`e`** to the Interactive Window
  - Use the **`arcpy.AddMessage()`** function to add the exception messages to the geoprocessing messages
- [ ] Check your script syntax and fix any errors that occur.
- [ ] In the Interactive Window, scroll down below any text and press the Enter key to display the Python prompt.
- [ ] Run your script.

4. Are the messages similar to the error messages in the previous step?

____

In the next step, you will work with ArcPy exceptions.

Step 3: Use arcpy.ExecuteError

In this step, you will work with the ArcPy ExecuteError error handler. This error handler will fire only when an arcpy error is generated.

- [ ] Comment out the Python Exception handler.
- [ ] Add the **`ExecuteError`** handler.
- [ ] Within the `arcpy.ExecuteError` handler, write code to print the following to the Interactive Window:
  - A message to indicate that a geoprocessing error occurred
  - The geoprocessing error messages
- [ ] Check your script syntax and fix any errors.
- [ ] Run your script.
- [ ] Review the messages in the Interactive Window.

5. Do the geoprocessing messages in the Interactive Window look any different than they did in the previous step?

____

6. If a Python exception had occurred, would the `arcpy.ExecuteError` handler code display any Python exceptions?

____

In the final step, you will work with the the `traceback` module.

Step 4: Use the Python traceback module

In this step, you will use a standard `except` block with the `traceback` module. In your script, the `traceback` will fire any time an exception is generated.

❑ Comment out the `ExecuteError` class exception handler.

❑ Add a standard **`except`** block.

- ☐ Within the `except` block, type the following code:

```
except:
    # Get the traceback object
    #
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]

    # Concatenate information together concerning the error into a message string
    #
    pymsg = "PYTHON ERRORS:\nTraceback info:\n" + tbinfo + \
            "\nError Info:\n" + str(sys.exc_info()[1])
    msgs = "ArcPy ERRORS:\n" + arcpy.GetMessages(2) + "\n"

    # Return python error messages for use in script tool or Python window
    #
    arcpy.AddError(pymsg)
    arcpy.AddError(msgs)

    # Print Python error messages for use in Python / Python window
    #
    print pymsg + "\n"
    print msgs
```

- ☐ Check your script syntax and fix any errors.

The Interactive Window has become cluttered with messages.

- ☐ Clear the Interactive Window. (Right-click within it, choose Select All, then right-click again and choose Cut).

- ☐ Press the Enter key to display the prompt.

- ☐ Run your script.

Several error messages print to the Interactive Window. You may need to scroll up to see all of them.

- ☐ Scroll up until you see the PYTHON ERRORS messages.

7. What are the two types of Python error messages that displayed?

____

8. Is the Error Info message content sufficient for you to possibly fix the problem?

____

❑ Scroll down to the ArcPy ERRORS.

9. Is the error message any different than the Error Info message?

____

10. When using the `traceback` module, do you need to add code to handle ArcPy messages?

____

In this exercise, you learned several techniques for handling both Python and ArcPy exceptions. The technique you use in your scripts will depend on how you want to handle the exceptions that may occur.

❑ Close your script.

Lesson review

1. Which exception handling technique (covered in this lesson) provides the most error detail?

2. Which exception handlers will handle only geoprocessing tool errors and warnings?

3. Write an except block that prints the error severity level geoprocessing messages to the Interactive Window.

Answers to Lesson 8 questions

Exercise 8: Working with exceptions

1. Why did the tool fail to execute?

   The parameters are not valid

2. What geoprocessing tool error has occurred?

   Error 000732: Clip Features: Dataset ClipBoundary does not exist or is not supported.

3. Based on the error messages, how might you fix the problem?

   Verify that the clipping feature does not exist. If it doesn't exist, create the feature class.
   Check the shapetype of the feature class to verify that the featue class stores polygon shapes.
   Open the feature class in ArcMap and visually check the features.

4. Are the messages similar to the error messages in the previous step?

   Yes, but there are fewer messages. Only the print and exception messages display.

5. Do the geoprocessing messages in the Interactive Window look any different than they did in the previous step?

   No, the geoprocessing messages are the same.

6. If a Python exception had occurred, would the `arcpy.ExecuteError` handler code display any Python exceptions?

   No, the `arcpy.ExecuteError` handler code only displays ArcPy Error exceptions.

7. What are the two types of Python error messages that displayed?

   Traceback and Error Info.

8. Is the Error Info message content sufficient for you to possibly fix the problem?

   Yes (because it displays the geoprocessing tool error messages).

9. Is the error message any different than the Error Info message?

   No, they are the same.

10. When using the `traceback` module, do you need to add code to handle ArcPy messages?

    No. (Python exception messages and geoprocessing messages will both display).

Lesson review

1. Which exception handling technique (covered in this lesson) provides the most error detail?

   The traceback module used in conjunction with the sys module can provide the most detail.

2. Which exception handlers will handle only geoprocessing tool errors and warnings?

   The arcpy.ExecuteError and arcpy.ExecuteWarning class exception handlers will raise an exception only if a geoprocessing tool error or warning is encountered.

3. Write an except block that prints the error severity level geoprocessing messages to the Interactive Window.

```
except:
    print arcpy.GetMessages(2)
```

9 Creating and updating geometry objects

Introduction

In many geoprocessing workflows, you will run tools that use coordinate and geometry information. There may be times when you do not necessarily want to go through the process of creating a temporary feature class to store features such as a clipping rectangle for the Clip tool. This would necessitate populating the feature class with a cursor, using the feature class in the geoprocessing task, and then deleting the feature class. Geometry objects can be used instead for the Clip tool's clipping layer input parameter to make the workflow simpler.

Geometry objects that can be created from scratch include Geometry, MultiPoint, Point, Polyline, and Polygon. These can be empty geometry objects with no coordinate values, or can be populated with coordinate pairs.

MultiPoint, Polyline and Polygon geometry objects use an array of coordinate pairs to construct the geometry object shape.

Learning objectives

After completing this lesson, you will be able to:

- Access ArcPy classes that create geometry objects
- Create and update features with geometry objects
- Use geometry objects in geoprocessing tasks

Key terms

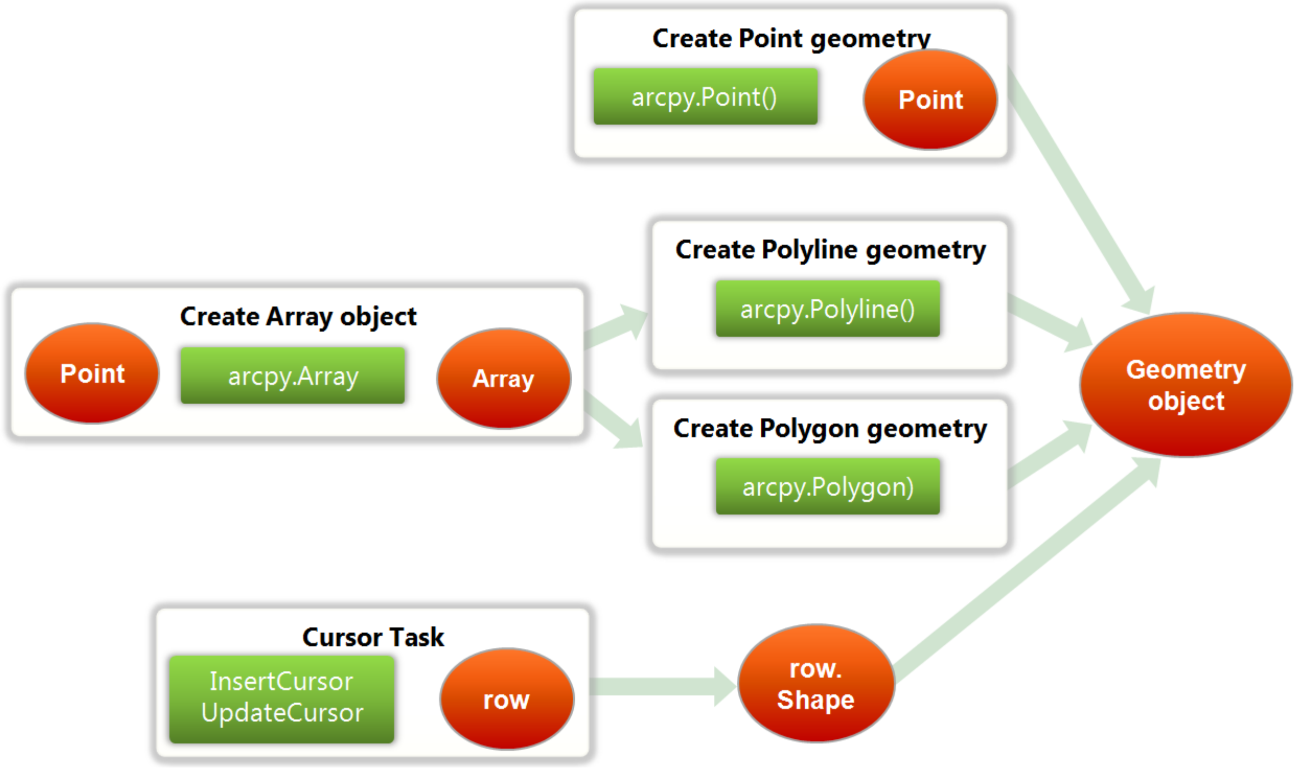
Geometry object : Used to define a spatial location and associated geometric shape.

Geometry list: When the output parameter of a geoprocessing tool is set to an empty Geometry object, the tool returns a list of Geometry objects.

Creating geometry objects

When performing Geoprocessing tasks, you may need to run a specific tool that uses coordinate or geometry information for input. Normally, you might create a temporary feature class to hold the geometry, populate the feature class with an Insert cursor, then use the feature class as input to a Geoprocessing tool.

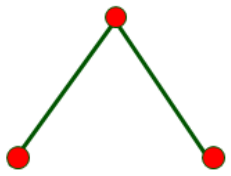
Geometry objects can be created from scratch for Geometry, Point, MultiPoint, Polygon, and Polyline ArcPy classes. The geometry objects can then be used for input to a Geoprocessing tool, which makes the workflow simpler than having to use a Feature Class or FeatureLayer for input.



Creating Point objects

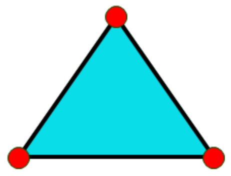
```
import arcpy
point = arcpy.Point(-98.36, 101.56)
print point.X
print point.Y
```

Creating Polyline objects



```
pnt1 = arcpy.Point(0,1)
pnt2 = arcpy.Point(1,2)
pnt3 = arcpy.Point(0,3)
array = arcpy.Array()
array.add(pnt1)
array.add(pnt2)
array.add(pnt3)
ln = arcpy.Polyline(array)
```

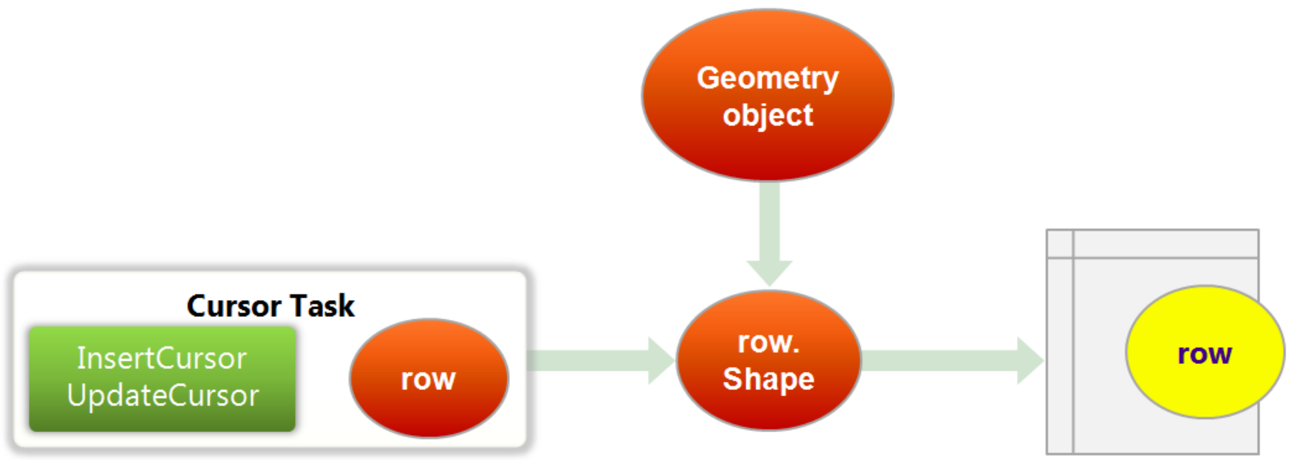
Creating Polygon objects



```
pnt1 = arcpy.Point(0,1)
pnt2 = arcpy.Point(1,2)
pnt3 = arcpy.Point(0,3)
array = arcpy.Array()
array.add(pnt1)
array.add(pnt2)
array.add(pnt3)
array.add(pnt1)
ply = arcpy.Polygon(array)
```

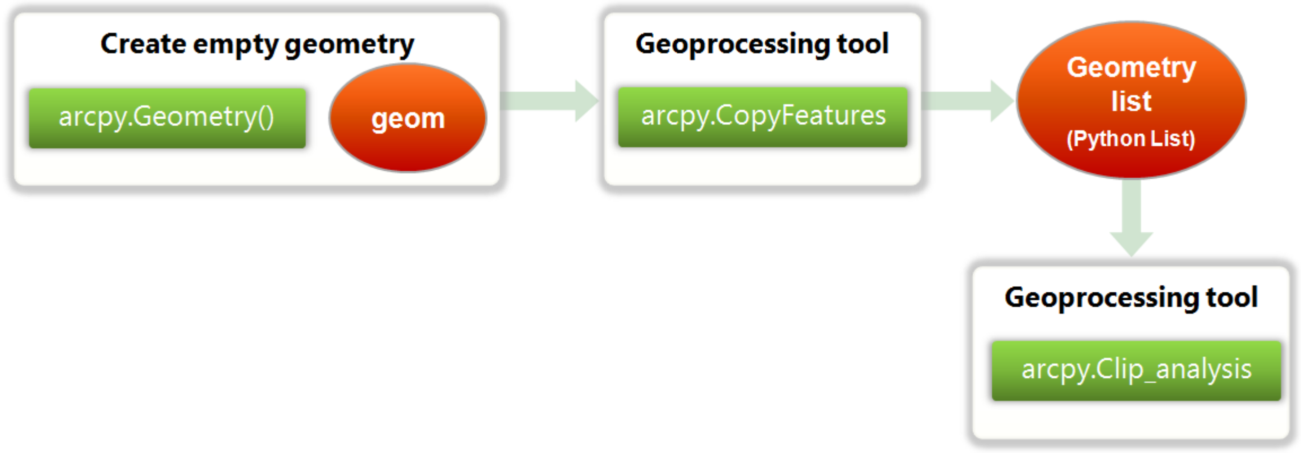
Creating and updating feature geometry

You can use cursors to insert new features into a feature class or update the existing features. When the cursor accesses the Shape field, geometry can be updated or created, as well as the feature attributes.



Working with a geometry list

Geometry objects can be created from a Geoprocessing tool by setting the output tool parameter to an empty Geometry object. When the tool runs, a Python list of geometry objects is returned. This list can be used to run statistics on field values, to populate a new feature class, or to take the place of a temporary feature class in your geoprocessing workflow.



The following example scripts use a geometry list.

Report buffered area

```
import arcpy
arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb "

g = arcpy.Geometry()
geomList = arcpy.Buffer("Freeways ", g, "10000 meters ")

area = 0
for geom in geomList:
    area += geom.area

print "Total area is: " + str(area)
# Freeways map units are in feet, convert sq ft to acres
print '%s: %f' % ("Total Acreage is", (area / 43560))
```

Buffered from geometry list

```
import arcpy
arcpy.env.workspace = "C:/Student/PYTH/Exercise09/Corvallis.gdb"

pnt= arcpy.Point()

featureList = []
coordList = [[1277000.0, 344000.0], [1283000.0, 344000.0],
             [1283000.0, 336000.0], [1277000.0, 336000.0]]

for coord in coordList:
    pnt.X = coord[0]
    pnt.Y = coord[1]
    pntGeometry = arcpy.PointGeometry(pnt)
    featureList.append(pntGeometry)

arcpy.Buffer_analysis(featureList, "BufferedLocations", "1000 feet")
```

Create geometry list from geoprocessing tool

```
# Create an empty Geometry object
g = arcpy.Geometry()

# Run the CopyFeatures tool, setting output to the geometry object.
# GeometryList is returned as a list of geometry objects.
geometryList = arcpy.CopyFeatures_management("c:/data/streets.shp", g)

# Walk through each geometry, totalling the length
length = 0
for geometry in geometryList:
    length += geometry.length

print "Total length: %f" % length
del g
```

Code samples

Point object

```
import arcpy

# Syntax: Point ({X}, {Y}, {M}, {Z}, {ID})
pnt = arcpy.Point(-98.8996, 29.58672)
```

Polyline object

```
import arcpy

# Syntax: Polyline (inputs, {spatialReference}, {hasZ}, {hasM})

# A list of coordinate pairs for a polyline
coordList = [[1,2], [2,4], [3,7]]

# Create empty Point and Array objects
point = arcpy.Point()
array = arcpy.Array()

for feature in coordList:
    # For each coordinate pair, set the x,y properties and add to the
    # Array object.
    point.X = feature[0]
    point.Y = feature[1]
    array.add(point)

# Create a Polyline object based on the array of points
polyline = arcpy.Polyline(array)

# Clear the array for future use
array.removeAll()
```

Polygon object

```
import arcpy

# Syntax: Polygon (inputs, {spatialReference}, {hasZ}, {hasM})

# A list of coordinate pairs for a polygon
coordList = [[1,2], [2,4], [3,7], [4,9]]

# Create empty Point and Array objects
point = arcpy.Point()
array = arcpy.Array()

for feature in coordList:
    # For each coordinate pair, set the x,y properties and add to the
    # Array object.
    for coordPair in feature:
        point.X = coordPair[0]
        point.Y = coordPair[1]
        array.add(point)

    # Add first point back to close the polygon
    array.add(array.getObject(0))

    # Create the object based on the array of points
    polyline = arcpy.Polygon(array)

    # Clear the array for future use
    array.removeAll()
```

Extent object

```
import arcpy

# Syntax:
# Extent ({XMin}, {YMin}, {XMax}, {YMax}, {ZMin}, {ZMax}, {MMin}, {MMax})

# Set the geoprocessing environment output extent to extent object
arcpy.env.extent = arcpy.Extent(-98.9967, 29.5455, -98.9975, 30.0157)
```

Using Geometry object in geoprocessing tool

```
# This script creates a Point geometry object
# that is passed to a geoprocessing tool

# The tool creates an in memory feature class containing
# a one mile buffer around the location of the Point

import arcpy

#Create the point to Buffer
pnt = arcpy.Point(6282633.845, 1838254.582)

#Create the Geometry object and pass the point
geom = arcpy.Geometry("Point",pnt)

#Buffer the point
bufferPnt = arcpy.Buffer_analysis(geom,"in_memory\\bufferPnt", "5280 Feet")
```

Exercise 9: Working with geometry objects

Estimated time: 30 minutes

Creation of geometry objects is quite easy. To create a new Point geometry, all you need is its x,y coordinates. For Polyline geometry, you construct the vertices of the geometry using Point objects that contain x,y coordinates, store the Point objects in an Array object, then pass the Array object to the Polyline class constructor. The same methodology applies to creating new Polygons, with the one additional step of adding the first point to the Array again at the end to close the polygon ring, then pass the Array to the Polygon class constructor.

Once the geometry object has been created, it can be used to create new features or modify existing features in feature classes. It can also be passed to a geoprocessing tool for both input or output parameters. In this exercise, you will create a Polygon geometry object and pass it to the Clip tool as the clipping layer. You will also create new geometry objects, populate them with coordinate pairs, and store them as new features in a feature class.

In this exercise, you will:

- Create Point, Polyline, and Polygon geometry objects
- Insert new features into a feature class using an InsertCursor and geometry objects
- Create a Polygon geometry object and pass it to the Clip tool as the clipping layer

Step 1: Creating Geometry objects

In this step, you will create new Point, Polyline, and Polygon geometry objects. First you will create a Point object.

- ❑ In the ArcGIS Desktop Help, navigate to:
  - Professional Library >
  - Geoprocessing >
  - The ArcPy site package >
  - Classes >
  - Point

- ❑ Review the following sections:
  - Summary
  - Discussion
  - Syntax
  - Code Sample: *Point example*

1. At a minimum, what is required to create a populated Point geometry object?

___

2. What code syntax would you write to create a new Point geometry object, where `X` is `2000` and `Y` is `1000`?

___

- ❑ Start PythonWin and create a new Python script.
- ❑ Save the script to the C:\Student\PYTH\Exercise09 folder as **CreateGeomObjects.py**.
- ❑ Using your answer to the previous question, create a new Point geometry object that reports its x,y coordinates to the Python Interactive Window.
- ❑ Check the code syntax, run the script, and verify that the coordinates print to the Interactive Window.
- ❑ Comment out your new code.
- ❑ Return to the ArcGIS Help and review the topics for the *Polyline* and *Polygon* classes.

Notice that you will need both Point and Array objects to create and populate Polyline and Polygon geometry objects.

- ❑ Return to PythonWin.

❑ Create a new Polyline geometry object based on the x,y coordinates provided below:

> **Note:** For guidance, you may want to refer to the *Polyline object* code sample that is provided on a preceding page in your workbook.

- Create an empty Point object and an empty Array object.
- Place the following x,y coordinate pairs in a Python list:

x-coordinate	y-coordinate
`100`	`200`
`200`	`400`
`300`	`700`
`600`	`800`
`500`	`700`

- Use a for loop to iterate through the Python list:
  - Extract each coordinate pair in the list and populate the X and Y properties of the Point object.
  - Add the point geometry to the array.
- Pass the array to the Polyline class constructor.

❑ Check for syntax errors in your script.

❑ Run your script.

❑ In the Interactive Window, write the following line of code to verify that the polyline was created successfully:

```
print polyline.pointCount
```

3. What value prints to the Interactive Window?

____

❑ Create a Polygon geometry object from the same list of coordinate pairs.

> **Note:** You do not need to add the first point again to close the polygon. When you pass the point array into `arcpy.Polygon()` it will automatically "rubber band" to the first point. However, it is good practice to add the first point to the end of the array. If you choose to add the first point to the end of the array, use this syntax:
>
> ```
> point.X = coordList[0][0]
> point.Y = coordList[0][1]
> ```

❑ Check your script syntax and run the script.

4. What is the value of the `pointCount` property for the polygon?

____

❑ Close your script.

Step 2: Using geometry objects to populate a feature class

In this step, you will create a new geometry object, then insert it into an existing feature class with an InsertCursor. You will also update the geometry for an existing feature class.

❑ In PythonWin, create a new Python script named **CreateFeatures.py** and save it to C:\Student\PYTH\Exercise09.

❑ Use these steps as a guide to write a script that inserts a new feature into a feature class:
- Create an Insert Cursor on the MajorAttractions feature class. (*Hint*: Set the workspace before creating the cursor.)
- Create an empty Point object and populate it with the following values:
  - **`X = 6284067.077`**
  - **`Y = 1840118.986`**
- Create a new row on the cursor and populate the row with these values:
  - **`Name = "Marthas Place"`**
  - **`CityNM = "San Diego"`**
  - **`Zip = 92109`**
- Populate the shape field with your Point object and insert the row into the cursor.
- Remember to delete the cursor to remove any locks on the feature class.

- ❑ Check your script.
- ❑ Start ArcMap.
- ❑ Open the SanDiegoMarina.mxd (from the Exercise09 folder).
- ❑ From the Bookmarks menu, choose Martha's Place.
- ❑ Open the Python window. Resize and move it to a good location.
- ❑ In the Python window, right-click and select Load.
- ❑ Browse to the Exercise09 folder, select the CreateFeatures.py, and click Open.

The script loads into the Python window.

- ❑ Press the Enter key to run the script.

It is good practice to refresh the view after the feature has been added.

- ❑ In the Python window, type in the following line of code:
  `arcpy.RefreshActiveView()`
- ❑ To verify that your script successfully added the new feature, use the Find tool to locate and zoom to Marthas Place.

Your next task is to update the location of Balboa Park to the new entrance that was opened last week.

- ❑ Go to the Balboa Park bookmark.

5. What are the coordinates for the Balboa Park point feature?
   (*Hint*: Use the Identify tool.)

   ____

- ❑ Return to PythonWin and open UpdateFeatures.py from the Exercise09 folder.
- ❑ In the script, modify the values of the Point's `X` and `Y` properties to the following:
  - **`X = 6285430.0`**
  - **`Y = 1844965.66`**

- ❑ Within the `for` loop:
  - Assign the Point object to the Shape field.
  - Update the cursor with the row object.
- ❑ Delete the cursor and refresh the view.
- ❑ Save your script and return to ArcMap.
- ❑ In the Python window, right-click and choose Clear All.
- ❑ Load your UpdateFeatures.py script and press Enter.
- ❑ Verify that the Balboa Park point feature location has been updated. (Use the Identify tool.)

Step 3: Use Geometry object in a script tool

When working with geoprocessing tools, you may need to create features in a temporary feature class and use that feature class as one of the inputs to the tool. This workflow typically requires that you create the empty feature class, populate the feature class, use the feature class in the geoprocessing tool, and then delete the temporary feature class.

You can use geometry objects as inputs to many of the geoprocessing tools. This can reduce the number of steps needed in the geoprocessing workflow and increase your efficiency.

In this step, you will create a Polygon geometry object to be used as input to the Clip_analysis tool. Then you will use the Clip tool to create a subset of downtown Corvallis streets.

- ❑ Return to PythonWin.
- ❑ Create a new Python script named **ClipGeom.py** and save it to your Exercise09 folder.
- ❑ Import arcpy and set the workspace to **`C:/Student/PYTH/Exercise09/Corvallis.gdb`**.
- ❑ Create a new empty Point object and a new empty Array object.

- ❑ Populate a polygon with the following x,y coordinates:

x-coordinate	y-coordinate
1277000.0	344000.0
1283000.0	344000.0
1283000.0	336000.0
1277000.0	336000.0

- ❑ Clip the StPaved feature class against the Polygon geometry object to create the DowntownStreets feature class.

Choose the correct Clip geoprocessing tool. (One of the Clip tools only works with rasters.)

- ❑ Check the syntax and run your script.

Now you will check the results.

- ❑ In ArcMap, open CorvallisStreets.mxd (from the Exercise09 folder).
- ❑ From the Geoprocessing menu, choose Geoprocessing Options.
- ❑ Confirm that the check box is enabled to Overwrite the outputs of geoprocessing operations.
- ❑ Add the DowntownStreets feature class to the map and verify that the Clip tool worked correctly.
- ❑ In the Table of Contents, right-click the DowntownStreets layer and choose Remove.
- ❑ Load the ClipGeom.py script into the Python window and press Enter to run the script.

DowntownStreets is added to the map.

- ❑ Close ArcMap and save your changes to the map.
- ❑ Close your scripts, then close PythonWin.

Lesson review

1. In order to create a Polygon geometry object, you must first create a ______ geometry object and an ______ object.

2. List some ways you might use geometry objects in your geoprocessing workflows.

____

____

____

Using Geometry objects in geoprocessing service

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Geometry_serverpopulation.py
# Summary: This script is designed to get population statistics
# for a 15min network analyst Drivetime polygon.
# Both the drivetime polygons and the population statistics
# are derived from 2 geoprocessing services made
# available from the sample server
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Import the required modules to run the script
import arcpy, time, sys

# Add the server toolbox through the url to the server toolbox.
# This sample will use the population summary geoprocessing
# service available on sample server
try:
    arcpy.ImportToolbox("http://sampleserver1.arcgisonline.com/ArcGIS/services;
                         Demographics/ESRI_Population_World")
    arcpy.ImportToolbox("http://sampleserver1.arcgisonline.com/ArcGIS/services;
                         Network/ESRI_DriveTime_US")

except:
    print "Toolbox not found or internet connection not there"
    print "Check the internet settings to the Sample server"
    sys.exit[1]

# Use the Point constructor to store the input geometry
pnt = arcpy.Point("-119.092","37.889")

# Add the point to a geometry object
geomObj = arcpy.Geometry("Point", pnt)

# Create a spatialReference
sf = arcpy.SpatialReference()
sf.factoryCode = 4326

# Validate what the output name will be
name = arcpy.CreateUniqueName("temporaryFC", "in_memory")
nameTemp = name.split("\\")[1]
```

```
# Create an in_memory featureclass
tempFC = arcpy.CreateFeatureclass_management("in_memory",nameTemp,
                                             "point","","","",sf)

# Append the geometry to the temporay featureclass
arcpy.Append_management(geomObj,tempFC.getOutput(0),"NO_TEST")

# Create a featureset object from tool parameter
networkFS = arcpy.GetParameterValue("CreateDriveTimePolygons",0)

# Load the featureclass into the featureset as input to the server tool
networkFS.load(tempFC.getOutput(0))

# Use the featureset to run the Drivetime polygons tool
netRes = arcpy.CreateDriveTimePolygons_ESRI_DriveTime_US(networkFS,"5 10 15")

# Wait for the Async process to finish.
# This is done because the gp service is asynchronous and the script
# needs to wait before using the results.
time.sleep(10)

# Test the status of the server tool.
# Status = 4 means the tool ran successfully.
if netRes.status != 4:
    print "Something happened with the network analysis"
    sys.exit[1]

# Get output from Drive time analysis tool available on sample server
networkOutput = netRes.getOutput(0)

# Use the get count tool to determine if there are features in the result
netResNum = arcpy.GetCount_management(networkOutput)

# Test the output to make sure there are features
if netResNum.getOutput(0) == 0:
    print "No network result detected"
    sys.exit[1]
else:
    popsummaryResult = arcpy.PopulationSummary_ESRI_Population_World(networkOutput)
    time.sleep(10)

# Determine the population output
populationResultNum = arcpy.GetCount_management(popsummaryResult.getOutput(0))
```

```
if populationResultNum == 0:
    print "Problem encountered with the population summary"
    sys.exit[1]
else:
    src = arcpy.SearchCursor(popsummaryResult.getOutput(0))
    row = src.next()
    totalPop = row.SUM
    print totalPop

# Release the cursors from memory
del src, row
```

Answers to Lesson 9 questions

Exercise 9: Working with geometry objects

1. At a minimum, what is required to create a populated Point geometry object?

   x,y coordinates

2. What code syntax would you write to create a new Point geometry object, where `X` is `2000` and `Y` is `1000`?

   `pnt = arcpy.Point(2000, 1000)`

3. What value prints to the Interactive Window?

   5

4. What is the value of the `pointCount` property for the polygon?

   6

5. What are the coordinates for the Balboa Park point feature?
   (*Hint*: Use the Identify tool.)

   6,285,474.996 and 1,844,795.379 feet

Lesson review

1. In order to create a Polygon geometry object, you must first create a **Point** geometry object and an **Array** object.

2. List some ways you might use geometry objects in your geoprocessing workflows.

   - **Use as input or output for geoprocessing tools**
   - **Create new features or update features**
   - **Create a Python list of geometry objects from the output of a geoprocessing tool to process further.**

Exercise solution

CreateGeomObjects.py

```
import arcpy

# Create new Point geometry object

# pnt = arcpy.Point(2000, 1000)
# print pnt.X
# print pnt.Y

# Create new Polyline geometry object

pnt = arcpy.Point()
array = arcpy.Array()
coordList = [[100,200], [200,400], [300,700], [600,800], [500,700]]

for coord in coordList:
    pnt.X = coord[0]
    pnt.Y = coord[1]
    array.add(pnt)

polyline = arcpy.Polyline(array)

# Create new Polygon geometry object

pnt = arcpy.Point()
array = arcpy.Array()
coordList = [[100,200], [200,400], [300,700], [600,800], [500,700]]

for coord in coordList:
    pnt.X = coord[0]
    pnt.Y = coord[1]
    array.add(pnt)

pnt.X = coordList[0][0]
pnt.Y = coordList[0][1]
polygon = arcpy.Polygon(array)
```

CreateFeatures.py

```
import arcpy
cur = arcpy.InsertCursor("MajorAttractions")
pnt = arcpy.Point()
pnt.X = 6284067.077
pnt.Y = 1840118.986
row = cur.newRow()
row.Name = "Marthas Place"
row.CityNM = "San Diego"
row.Zip = 92109
row.Shape = pnt
cur.insertRow(row)
del cur
```

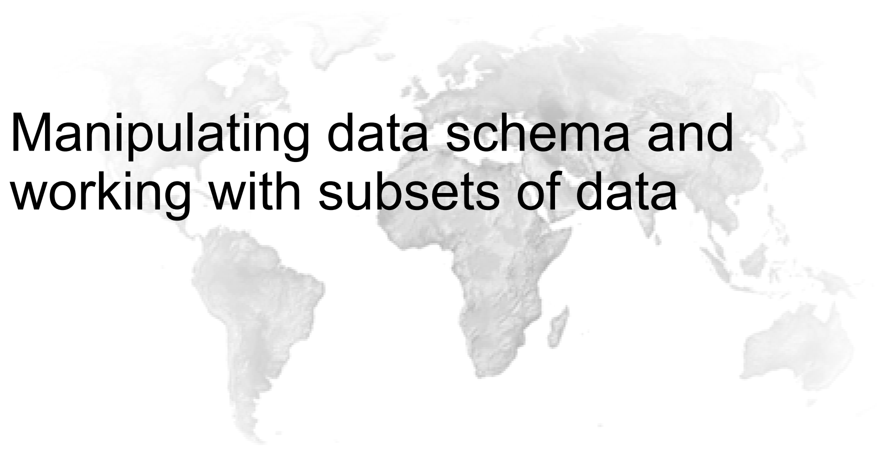
UpdateFeatures.py

```
import arcpy
arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"
cur = arcpy.UpdateCursor("MajorAttractions", "NAME = 'BALBOA PARK'")
pnt = arcpy.Point()
pnt.X = 6285430.0
pnt.Y = 1844965.66

for row in cur:
    row.Shape = pnt
    cur.updateRow(row)

del cur
arcpy.RefreshActiveView()
```

ClipGeom.py

```
import arcpy
arcpy.env.workspace = "C:/Student/PYTH/Exercise09/Corvallis.gdb"

pnt = arcpy.Point()
ary = arcpy.Array()

coordList = [[1277000.0, 344000.0], [1283000.0, 344000.0],
             [1283000.0, 336000.0], [1277000.0, 336000.0]]
```

```
for coord in coordList:
    pnt.X = coord[0]
    pnt.Y = coord[1]
    ary.add(pnt)

pnt.X = coordList[0][0]
pnt.Y = coordList[0][1]
ary.add(pnt)
clipPoly = arcpy.Polygon(ary)

arcpy.Clip_analysis("StPaved", clipPoly, "DowntownStreets")
```



10 Manipulating data schema and working with subsets of data

Introduction

Automating the process of creating new geodatabases, feature classes, and tables can be advantageous when designing, testing, and implementing a new GIS system. You can easily make changes to schema, field names, and other components as needed.

What if you need to create a new feature class as a subset of a larger feature class? How about if field names need to change to fit a new requirement? What are your options in this regard? You have a couple of different paths you can choose to make the changes:

- Use ArcCatalog and make the changes by hand.
- Create a model or script and automate the process.
- Create subsets of data by making a feature layer or table view, then copy the features/rows to a new feature class or table.

The Geoprocessing tools MakeFeatureLayer and MakeTableView both accept a SQL expression and a FieldInfo object as their parameters. Once the FeatureLayer or TableView object is created in memory, the CopyFeatures tool can be used to write the FeatureLayer features to a feature class and/or the CopyRows tool can write the TableView rows to a new table.

In the case of just changing a field, you can create a FieldInfo object detailing the change, create a FeatureLayer or TableView object using the FieldInfo object, then copy the features/rows to a new feature class/table.

Learning objectives

After completing this lesson, you will be able to:

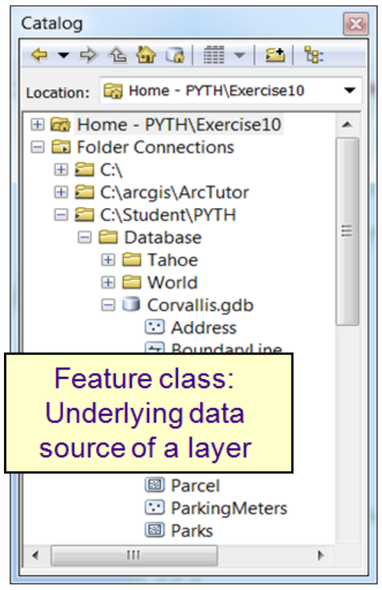
- Determine whether to use a FeatureLayer or FeatureClass in a tool
- Determine whether to use a Table or TableView in a tool
- Create a FeatureLayer that uses a FieldInfo object
- Construct appropriate field delimiters for a SQL expression based on the workspace type

Key terms

- **FeatureLayer**: An in-memory spatial representation of the data in a FeatureClass
- **FeatureClass**: A table containing an attribute field that stores the shape of a feature
- **Table**: A storage container for rows that contain attribute fields to store values
- **TableView**: An in-memory representation of the data in a Table
- **FieldInfo object**: An object that provides methods and properties for working with fields

Feature layer and table view

Feature class vs. feature layer



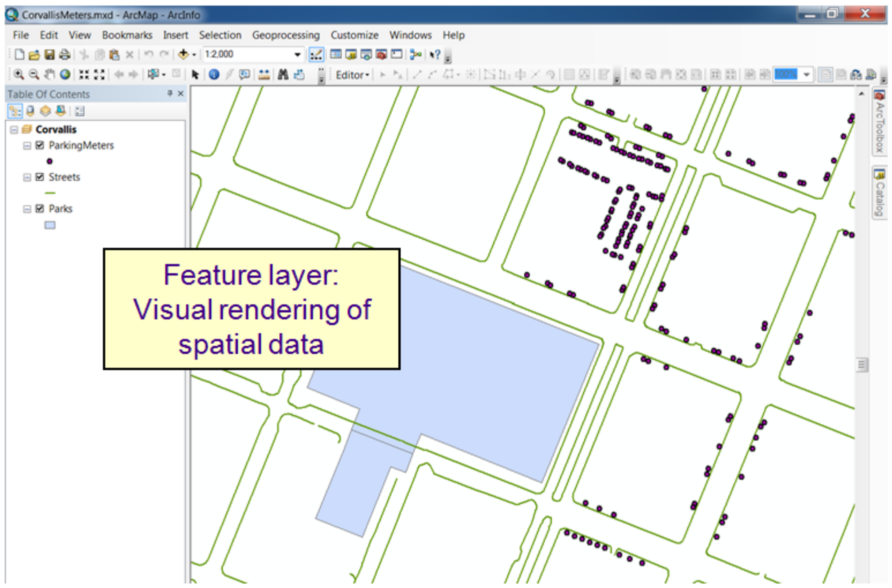
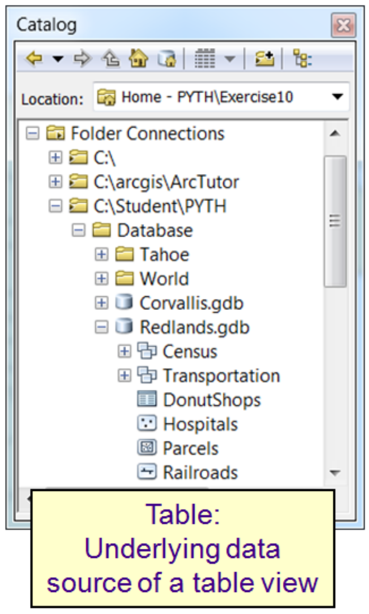
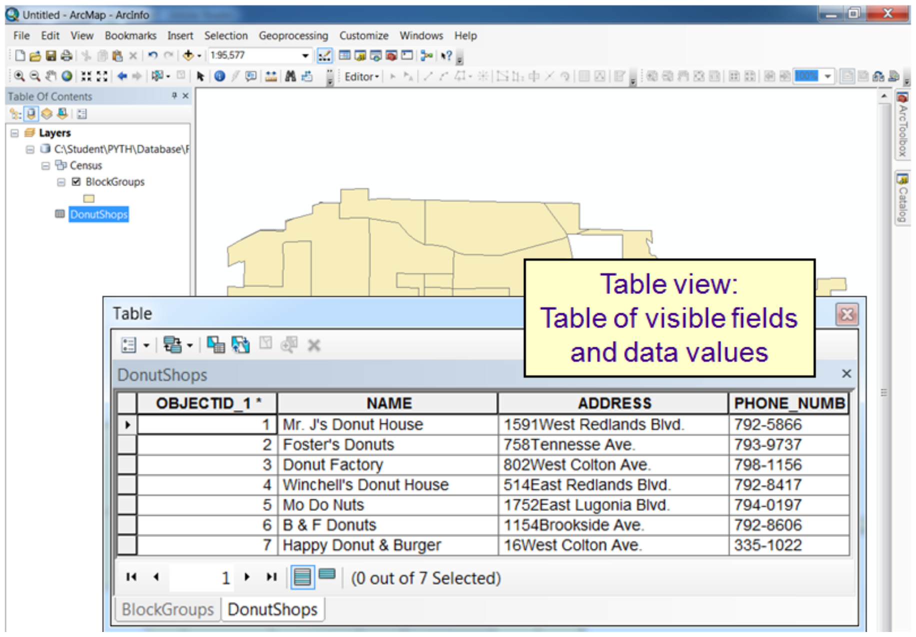


Table vs. table view





OBJECTID_1 *	NAME	ADDRESS	PHONE_NUMB
1	Mr. J's Donut House	1591West Redlands Blvd.	792-5866
2	Foster's Donuts	758Tennesse Ave.	793-9737
3	Donut Factory	802West Colton Ave.	798-1156
4	Winchell's Donut House	514East Redlands Blvd.	792-8417
5	Mo Do Nuts	1752East Lugonia Blvd.	794-0197
6	B & F Donuts	1154Brookside Ave.	792-8606
7	Happy Donut & Burger	16West Colton Ave.	335-1022

Tools that create and manage feature layers and table views

Many geoprocessing tools require a Feature Layer for input. Some of these same tools will not accept a Feature Class as input; for example, Select Layer by Attribute requires a feature layer or table view.

In the ArcGIS Desktop Help, expand:

- Professional Library >
- Geoprocessing >
- Geoprocessing tool reference >
- Data Management toolbox >
- Layers and Table Views toolset

Refer to the following help topics to complete the table below and to answer the questions:

- *An overview of the Layers and Table Views toolset*
- Layers and Table Views toolset concepts > *Creating and using layer selections*

1.

Which tools create or manage feature layers and table views?

2. Of the tools that you listed in the table:
   - Do any create an output feature layer or table view?
   - Do all these tools require either a feature layer or table view for input?

____

____

3. What is the main benefit of using a feature layer as input to a geoprocessing tool?

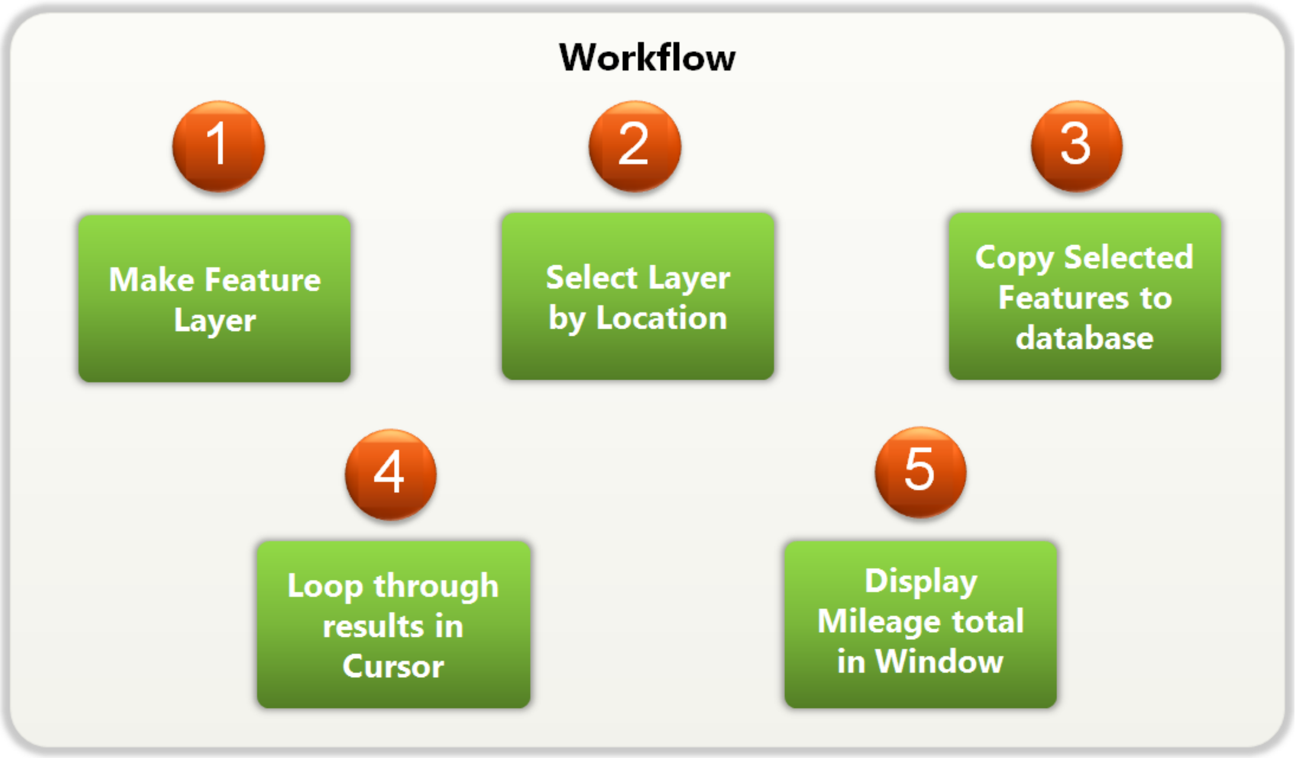
____

4. What geoprocessing tool can you use to make a feature layer permanent?

____

Workflow problem

Report on road mileage in commissioner district


Workflow
1
Make Feature Layer
2
Select Layer by Location
3
Copy Selected Features to database
4
Loop through results in Cursor
5
Display Mileage total in Window

Creating a FeatureLayer object

Use the MakeFeatureLayer tool to create a FeatureLayer object. Provide the tool with either a feature class or another feature layer to create a new in-memory feature layer.

Create a feature layer and copy subset to a new feature class

```
# Import arcpy site package and set the workspace
import arcpy
arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"

# Create a Feature Layer from the MajorAttractions feature class
# A SQL expression will be applied to filter results to only include
# features with a valid established date prior to 1956
arcpy.MakeFeatureLayer_management("MajorAttractions",
                                  "SelMajorAttractions",
                                  "Estab > 0 and Estab < 1956")

# Store the Feature Layer feature count in a variable
# and print to the Interactive Window.
featCount = arcpy.GetCount_management("SelMajorAttractions")
print "Number of features in feature layer: " + str(featCount)

# Make a permanant feature class from the Feature Layer.
arcpy.CopyFeatures_management("SelMajorAttractions",
                              "HistoricAttractions")
```

Using the FieldInfo object

Use the FieldInfo object to define field properties such as a new name, field visibility, and setting split rules.

Use a FieldInfo object in the MakeFeatureLayer tool

```
# Import arcpy site package and set the workspace
import arcpy
arcpy.env.workspace = "C:/Student/PYTH/Database/Corvallis.gdb"

# Create a FieldInfo object
fldInfo = arcpy.FieldInfo()

# The PARK_NAME field name is to be changed to NAME.
# Add the change to the FieldInfo object
fldInfo.addField("PARK_NAME", "NAME", "VISIBLE", "")

# Apply the FieldInfo object to the MakeFeatureLayer tool
# to define the new field name.
# A SQL expression will also be applied to filter results
# that include features with an area greater than 200000.
arcpy.MakeFeatureLayer_management("Parks", "ParksLyr",
                                  "Shape_Area > 200000", "", fldInfo )

# Make a permanant feature class from the Feature Layer.
arcpy.CopyFeatures_management("ParksLyr", "LargeParks")
```

Using field delimiters in a SQL query

When creating the SQL expression for the MakeFeatureLayer and MakeTableView tools, you can use an optional SQL expression to subset the input features. To properly format the SQL expression, you can apply the AddFieldDelimiters ArcPy function to the SQL expression. The function works with the current workspace environment setting or a specified workspace to determine and apply the correct field delimiters for the tool.

Syntax

`arcpy.AddFieldDelimiters(datasource, field)`

Code sample

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Construct a properly delimited
# SQL expression based on the workspace
#
#~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Import the ArcPy site package and set the workspace
import arcpy
arcpy.env.workspace = "C:/Student/PYTH/Database/SanDiego.gdb"

# Setup some initial variables for field and value
fldName = "TYPE"
value = "Maritime"

# Construct properly delimited field name
newFld = arcpy.AddFieldDelimiters(arcpy.env.workspace, fldName)

# Construct SQL expression to apply to Feature Layer
sqlExp = newFld + " = " + "'" + value + "'"

# Feature Layer will contain only maritime climate polygons
arcpy.MakeFeatureLayer_management("Climate",
                                  "MaritimeClimate", sqlExp)

# Print feature counts to Interactive Window
featClassCount = arcpy.GetCount_management("Climate")
featLyrCount = arcpy.GetCount_management("MaritimeClimate")
print "Climate feature class contains " + str(featClassCount) + " features"
print "Maritime climate feature layer contains " + str(featLyrCount) + " features"
```

Syntax depends on type of workspace

SELECT * FROM <table> WHERE:
fd NAME fd = 'China'

Workspace	Field delimiter	Example
• Shapefile • File geodatabase • CAD • Coverage	" "	"NAME" = 'China'
• Personal geodatabase	[ ]	[NAME] = 'China'
• Multiuser geodatabase	nothing	NAME = 'China'

Exercise 10: Working with subsets of data

Estimated time: 30 minutes

There are times when you may need to create a subset of features from a feature class or layer in your map, make some changes to the fields in the subset, and copy them to a new feature class for more processing. This exercise will cover a sample workflow to accomplish this task.

In this exercise, you will write a Python script that will:

- Construct a SQL expression with proper field delimiters
- Create a FeatureLayer that uses the SQL expression to create a subset of features
- Create a second FeatureLayer for a spatial selection
- Perform a spatial selection using a specified distance
- Apply a FieldInfo object on the result of the spatial selection to hide fields
- Use the CopyFeatures geoprocessing tool to create a new FeatureClass from the subset

Step 1: Create a subset of features

In this step, you will write a script that creates a FeatureLayer from a feature class. The MakeFeatureLayer tool will use a SQL expression to create a subset of features contained within the FeatureLayer.

The City of Corvallis will be holding a special fundraiser at Central Park. Your task is to identify the parking meters that are within a 500-foot distance of Central Park. The meters will be programmed to charge a special reduced rate for the event.

- ❑ Open PythonWin and create a new Python script.
- ❑ Save the script to your Exercise10 folder with the name **CreateCentralParkMeters.py**.
- ❑ Import the ArcPy site package and set the workspace environment setting to **`"C:/Student/PYTH/Database/Corvallis.gdb"`**

In order to create a FeatureLayer that will contain only the Central Park feature, you will need to construct a SQL expression with the correct delimiters around the field name.

- ❑ Using the **`arcpy.AddFieldDelimiters`** function, create field delimiters around the **`PARK_NAME`** field in the **`Parks`** feature class and assign it to the **`nameFld`** variable.
- ❑ Assign the value **`"Central Park"`** to the **`value`** variable.

- ❑ Construct the SQL expression using the following code:
  `sqlExp = nameFld + " = " + "'" + value + "'"`

> ⚠ **There is a single quotation mark surrounded by double quotation marks around the `value` variable. In a SQL expression, the value being evaluated is always surrounded by single quotes, for example:**
> **"STREET_NAME" = 'MAIN St'**

- ❑ Write code to create a FeatureLayer that will contain the Central Park feature.
  - `in_features`: **`"Parks"`**
  - `out_layer`: **`"Central Park"`**
  - `where_clause`: **`sqlExp`**

- ❑ Check the syntax of your script. Do not run the script yet.

Step 2: Perform analysis on FeatureLayer

In this step, you will create a FeatureLayer for the parking meters, select all the parking meters within 500 feet of Central Park and copy the meters to a new feature class.

There are some schema changes that you should make to the parking meters before you start working with the features. You can make the changes with a FieldInfo object and apply it to the MakeFeatureLayer tool.

- ❑ Create an empty FieldInfo object and name it **`fldInfo`**.

- ❑ Using the **`addField`** method on **`fldInfo`**, write code to make these schema changes:
  - "RECCFLAG", "", "HIDDEN", ""
  - "RECAFLAG", "", "HIDDEN", ""
  - "METER_NUM", "METERNUM", "VISIBLE, ""

- ❑ Create a new FeatureLayer for the Parking Meters. You can use a pair of double-quotes to skip a parameter.
  - in_features = **"ParkingMeters"**
  - out_layer = **"METERS"**
  - where_clause = **""**
  - workspace = **""**
  - field_info = **fldInfo**

Now that you have a FeatureLayer containing parking meters with the desired schema changes, you are ready to select all the parking meters that are within 500 feet of the Central Park feature.

- ❑ In the ArcGIS Desktop Help, review the parameters for the SelectLayerByLocation tool.

1. What overlap type and selection type would you use?

____

____

- ❑ Using the `SelectLayerByLocation` tool, write code to select parking meters that are within 500 feet of Central Park.
  - in_layer = **"METERS"**
  - overlap_type = use your answer to the previous question
  - select_features = **"Central Park"**
  - search_distance = **"500 feet"**
  - selection_type = use your answer to the previous question

Your final step is to create a new feature class from the selected meters.

- ❑ Write code to copy the selected meters to a new feature class.
  - in_features = **"METERS"**
  - out_feature_class = **"CentralParkMeters"**
- ❑ Check your code syntax and run the script.
- ❑ To check the results of your script, use the `GetCount_management` tool. You can also open ArcMap, add the feature class, and open the attribute table.

2. How many features are stored in the CentralParkMeters feature class?

____

Lesson review

1. What are two geoprocessing tools that must use a feature layer to make selections?

2. Using a FieldInfo object, what kinds of schema changes can you make on a field?

3. If you set a field as hidden in a FieldInfo object, the field will still be available in the Feature Layer.

   a. True

   b. False

Answers to Lesson 10 questions

Tools that create and manage feature layers and table views

1.

Which tools create or manage feature layers and table views?
Make Feature Layer
Make Query Table
Make Table View
Make XY Event Layer
Save To Layer File
Select Layer By Attribute
Select Layer By Location

2. Of the tools that you listed in the table:
   - Do any create an output feature layer or table view?
   - Do all these tools require either a feature layer or table view for input?

     - **Yes: Make Feature Layer, Make Table View, and Make Query Table.**
     - **No, only some do: Select Layer By Attribute, Select Layer By Location, and Save To Layer File.**

3. What is the main benefit of using a feature layer as input to a geoprocessing tool?

   Only the selected features will be used.

4. What geoprocessing tool can you use to make a feature layer permanent?

   The CopyFeatures_management tool.

Exercise 10: Working with subsets of data

1. What overlap type and selection type would you use?

   Overlap type: WITHIN_A_DISTANCE
   Selection type: NEW_SELECTION

2. How many features are stored in the CentralParkMeters feature class?

   176

Lesson review

1. What are two geoprocessing tools that must use a feature layer to make selections?

   SelectLayerByAttribute, SelectLayerByLocation

2. Using a FieldInfo object, what kinds of schema changes can you make on a field?

   - **Rename a field**
   - **Make the field hidden or visible**
   - **Set a split rule**

3. If you set a field as hidden in a FieldInfo object, the field will still be available in the Feature Layer.

   b. False

Exercise solution

CreateCentralParkMeters.py

```
# Import arcPy site package and set the workspace environment setting
import arcpy
arcpy.env.workspace = "C:/Student/PYTH/Database/Corvallis.gdb"

# Create a field-delimited object, construct a SQL expression
# and create a feature layer using the SQL expression
nameFld = arcpy.AddFieldDelimiters("Parks", "PARK_NAME")
value = "Central Park"
sqlExp = nameFld + " = " + "'" + value + "'"
arcpy.MakeFeatureLayer_management("Parks", "Central Park", sqlExp)

# Create and populate a FieldInfo object, then create a feature layer
# using the FieldInfo object
fldInfo = arcpy.FieldInfo()
fldInfo.addField("RECCFLAG", "", "HIDDEN", "")
fldInfo.addField("RECAFLAG", "", "HIDDEN", "")
fldInfo.addField("METER_NUM", "METERNUM", "VISIBLE", "")
arcpy.MakeFeatureLayer_management("ParkingMeters", "METERS",
    "", "", fldInfo)

# Perform a spatial selection on the parking meters to find all that
# are within a distance of 500 meters to the Central Park feature
# Copy the results to a new feature class
arcpy.SelectLayerByLocation_management("METERS", "WITHIN_A_DISTANCE",
    "Central Park", "500 feet", "NEW_SELECTION")
arcpy.CopyFeatures_management("METERS", "CentralParkMeters")

# Verify the results
print arcpy.GetCount_management("CentralParkMeters")
```

11 Automating map production with ArcPy mapping module

Introduction

Making maps of your data is a key function of ArcGIS. The map may illustrate the results of performing analysis, show certain patterns or relationships in your data, or simply convey information about your data in an organized way.

This lesson will focus on automating the process of creating map books in ArcGIS. There are two different ways to organize the layout of the map books. You can place a grid of polygons over the extent of the features, which is called a grid index series, or the polygons can follow a single or group of linear features, which is called a strip map index series.

There are many different types of map books that you can create in ArcGIS.

- *Simple reference series map book*: A set of map pages that use a single layout for a set of map extents
- *Reference series map book*: Contains a set of map pages, including a title page, overview/ locator map, ancillary information, and contact information
- *Thematic map book*: Each map in the map series shows unique thematic maps of a single location
- *Reference map book with insets*: A map series with inset maps for more detail, such as for a densely populated area

Learning objectives

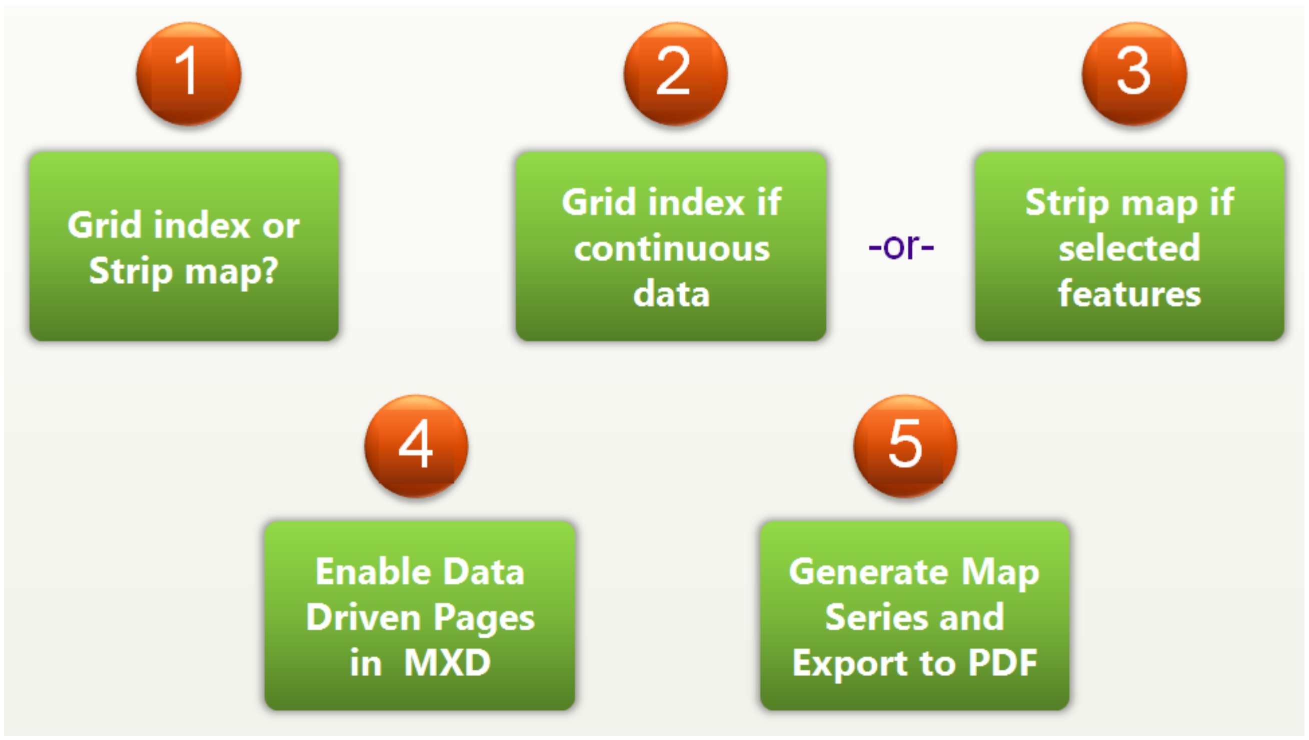
After completing this lesson, you will be able to:

- Set up an MXD for printing a map series
- Work with Data Driven Pages
- Create and export a map series

Key terms

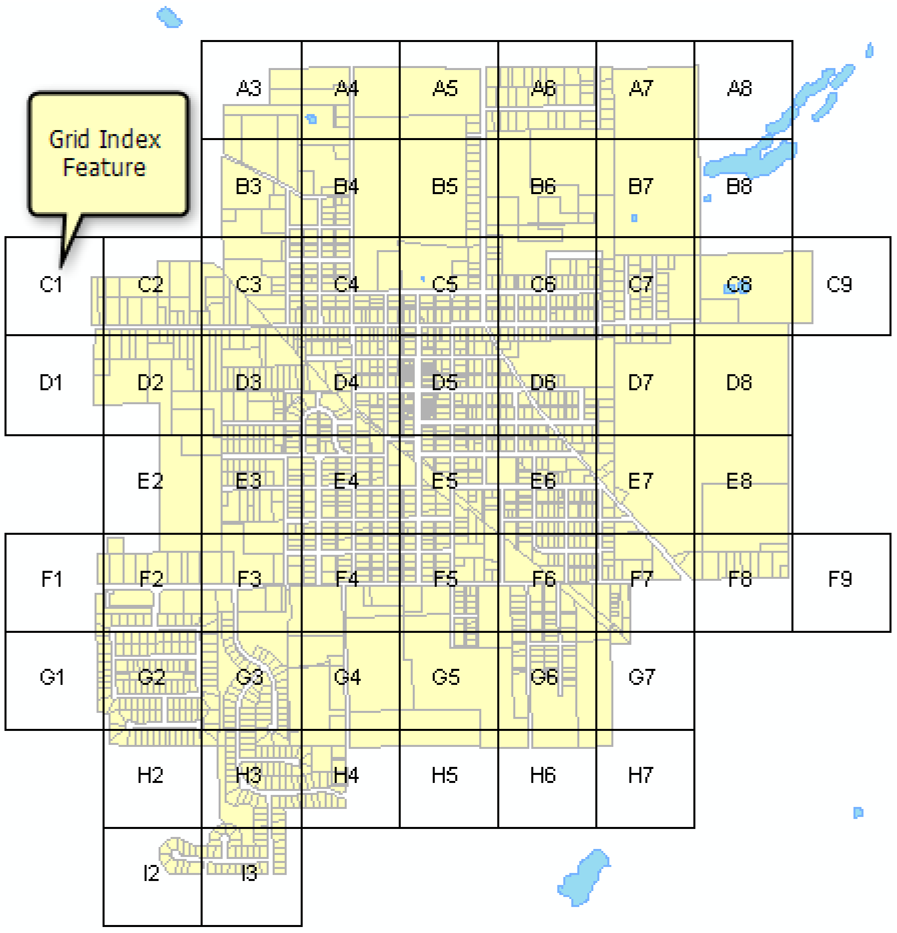
- **Map series**: A set of map pages.
- **Map book**: A collection of pages which includes a map series. May contain Title page, Table of Contents, tabular information, and other content.
- **Index series**: A layer of rectangular polygons that define a set of map extents. May also be referred to as tiles, sections, or areas of interest (AOI).
- **Data Driven Pages**: Provides the ability to generate a set of pages from a single layout by iterating over an index series. Data Driven Pages are created and customized in the ArcMap Layout View using the Setup Data Driven Pages dialog.

Setting up an MXD for a map book: Workflow


1
Grid index or Strip map?
2
Grid index if continuous data
-or-
3
Strip map if selected features
4
Enable Data Driven Pages in MXD
5
Generate Map Series and Export to PDF

Grid index map series

A grid or fishnet of polygons is placed over the extent of the features and used to generate the map pages.



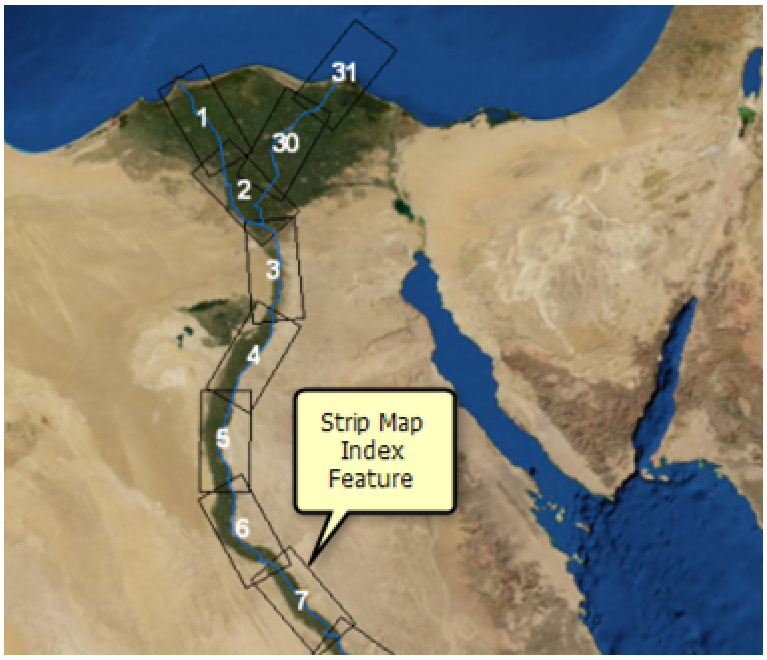
Create a grid index series

```
import arcpy
from arcpy import env
env.workspace = "C:/Student/PYTH/Database/Corvallis.gdb"

# Create grid index using the entire extent of input features,
# specifying the grid size in map units, and start page numbering at 3.
arcpy.GridIndexFeatures_cartography("gridIndexFeatures", "Boundary",
                                    "NO_INTERSECTFEATURE",  "", "",
                                    "1000 meters", "1000 meters", "3")

# Populate adjacent map name fields
arcpy.CalculateAdjacentFields_cartography("gridIndexFeatures", "PageName")
```

```
# Add fields for a reference grid (optional)
arcpy.CalculateCentralMeridianAndParallels("gridIndexFeatures",
                                            "CentralMeridian", 0.25)

# Calculate a UTM zone for the map series (optional)
arcpy.CalculateUTMZone_cartography("gridIndexFeatures", "UTM_Zone")

# Calculate a Grid Convergence Angle for the map series (optional)
arcpy.CalculateGridConvergenceAngle_cartography("gridIndexFeatures",
                                                 "angle", "GEOGRAPHIC")
```

Strip map index series

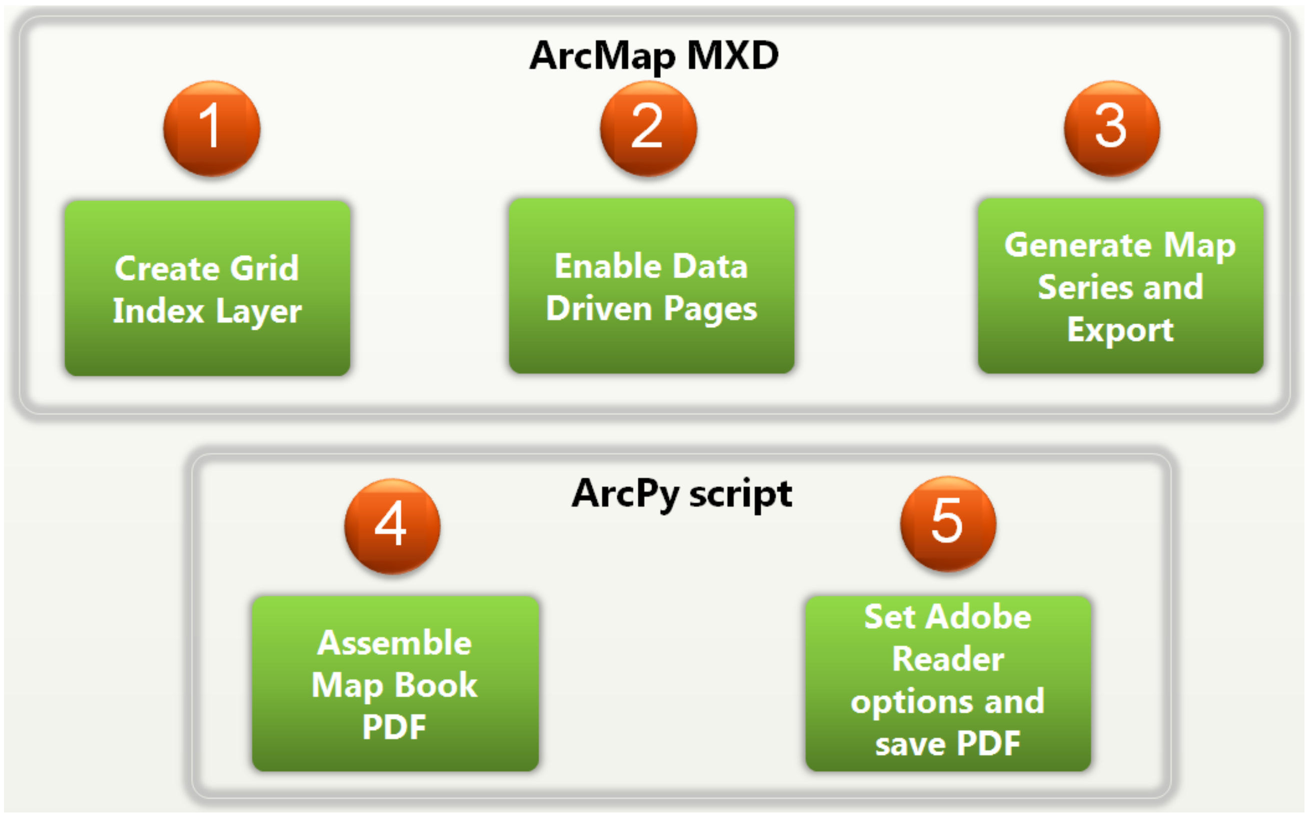
A series of rectangular polygons is placed over a group of linear features and used to generate the map pages.



Create a strip map series

```
# Import system modules
import arcpy
from arcpy import env

# Set environment settings
arcpy.env.workspace = C:/Student/PYTH/Database/Corvallis.gdb"

# Set local variables
inFeatures = "Rail100"
outFeatureClass = "stripIndexFeatures"
usePageUnit = "USEPAGEUNIT"
scale = "500000"
lenA = "7 Inches"
lenP = "5 Inches"

# Execute StripMapIndexFeatures
arcpy.StripMapIndexFeatures_cartography(inFeatures, outFeatureClass,
                                        usePageUnit, scale, lenA, lenP)
```

Creating a reference mapbook: Workflow



Create and export a reference map book

Create the grid index feature layer

Add fields to store adjacent map names

Enable and set up Data Driven Pages in ArcMap

Export Data Driven Pages map series to PDF

Create new map book PDF

Append title page to map book PDF

Append map series to map book PDF

Append contact info page to map book PDF

Set up initial view settings in map book PDF

Save and close map book PDF

Exercise 11: Creating a map series book

Estimated time: 45 minutes

In this exercise, you will write a script that creates a reference series map book.

Your script will:

- Create the map book PDF
- Append the title page to the map book PDF
- Export and append the grid index map series to the map book PDF
- Append the Contacts Info page to the map book PDF
- Save the map book PDF and view it

Step 1: Create the MapBook output document

In this step, you will write a Python script to create a map book and compile its contents.

- ❑ If necessary, open PythonWin.
- ❑ From your ..\PYTH\Exercise11 folder, open the script PublishMapBook.py and save it as **MyPublishMapBook.py**.
- ❑ Review the comments in the script to grasp a basic understanding of the processing sequence.

In the script, you will write code below the relevant comments.

- ❑ Below the comment `# Import ArcPy and os modules`:
  - Import the arcpy and os modules.
- ❑ Below the relevant comment:
  - Set the current workspace to your Exercise11 folder.
- ❑ Below the relevant comment, create variables for the output path and PDF file name:
  - Set a variable named **`outDir`** to your **`Exercise11`** folder.
  - Set a variable named **`finalpdf_filename`** to **`outDir + r"\FinalMapBook.pdf"`**.
- ❑ Check whether the multi-page PDF already exists, and if it does, remove it:

```
if os.path.exists(outDir + r"\MapPages.pdf"):
       os.remove(outDir + r"\MapPages.pdf")
```

- ❑ Check whether the map book PDF already exists, and if it does, remove it:

```
if os.path.exists(finalpdf_filename):
       os.remove(finalpdf_filename)
```

- ❑ Create the PDF for the map book:
  - Call the **`arcpy.mapping.PDFDocumentCreate`** function.
  - Pass in **`finalpdf_filename`** as the argument.
  - Store the result in a variable named **`finalPDF`**.

1. Which variable can you use whenever you write code that includes the path to your Exercise11 folder?

____

- ❑ Append the title page to `finalPDF`:
  - Call the **`appendPages()`** function on **`finalPDF`**.
  - For the argument, pass in the path to **`PlainsViewTitlePage.pdf`** in your Exercise11 folder.

Best practice:
- Save your work often.
- Click Check to run the Tab Nanny.
- Resolve any syntax errors.

Step 2: Export the Data Driven pages

You will use the following sequence to create the pages from the grid index and export the pages to PDF:

1. Create a MapDocument object document that points to the specified map document (*.mxd).
2. Export the current data driven pages to a temporary PDF.
3. Append the exported page to the map book PDF.
4. Delete the temporary PDF.

Note: To learn more about exporting data driven pages, go to the ArcGIS Desktop Help and navigate to:

- Professional Library >
- Mapping and Visualization >
- Automating map workflows >
- Data driven pages >
- Exporting Data Driven Pages

❑ Create a `MapDocument` object (*Hint*: Use the variable that includes the path to your Exercise11 folder):

- Call the **`arcpy.mapping.MapDocument()`** function.
- For the argument, pass in the path to **`PlainsView.mxd`**.
- Store the result in a variable named **`mxd`**.

Next, you will export the map document that contains the data driven pages to a multi-page PDF.

❑ Export the Data Driven Page MXD to a multi-page PDF:

- Set a variable named **`ddp`** to the **`dataDrivenPages`** property of **`mxd`**.
- Print a message to tell the user that the export process is beginning.
- Export the data driven pages to the PDF:
  - Call the **`exportToPDF()`** function on **`ddp`**.
  - Pass in the path to **`MapPages.pdf`**.

Step 3: Compile the final map book

Now that you have exported the data driven pages, you are ready to create the final output.

❑ Append the multi-page PDF to finalPDF:

- Call the **`appendPages()`** function on **`finalPDf`**.
- For the argument, pass in the ouput path to **`MapPages.pdf`**.

❑ Append the Contact page to the PDF:

- Call the **`appendPages()`** function on **`finalPDf`**.
- For the argument, pass in the output path to **`PlainsViewContactPage.pdf`**.

- ❑ Update the properties for viewing in Adobe Reader and save the PDF:
  - Call the **`updateDocProperties()`** function on **`finalPDF`**.
  - Pass in the following arguments:
    - **`pdf_open_view = "USE_THUMBS"`**

      Note: This argument determines how thumbnails are handled.

    - **`pdf_layout = "SINGLE_PAGE"`**
  - Call the **`saveAndClose()`** function on **`finalPDF`** (with no arguments).
- ❑ Delete the references to **`mxd`** and **`finalPDF`**.
- ❑ Print a message to tell the user that the creation of the map book is complete.
- ❑ Check your syntax and run your script.
- ❑ Open your FinalMapBook PDF from your ..\PYTH\Exercise11 folder.

2. Are the pages printed from Data View or Layout View? How can you tell?

____

Note: You can make your script dynamic by using the `arcpy.GetParameterAsText()` function. Then you could add it as a script tool to a custom toolbar and run the script in ArcMap.

- ❑ Close all open files and windows.

Lesson review

1. Can you enable Data Driven Pages from a script?

2. What can you do to make map book-creation more automated?

Answers to Lesson 11 questions

Exercise 11: Creating a map series book

1. Which variable can you use whenever you write code that includes the path to your Exercise11 folder?

   outDir

2. Are the pages printed from Data View or Layout View? How can you tell?

   Layout View. The map elements such as scale bar and north arrow display

Lesson review

1. Can you enable Data Driven Pages from a script?

   No. Data Driven Pages must be enabled in ArcMap.

2. What can you do to make map book-creation more automated?

   1. **Replace hardcoded variables with `arcpy.GetParameterAsText()`.**
   2. **Add scripts to custom toolbar as script tools and run on MXDs.**

Exercise solution

MyPublishMapBook.py

```
# PublishMapBook.py
# Author:  <Your Name>
# Date: <Today>
# Purpose:  Create map book pdf, output Data Driven Pages series from
# mxd, assemble map book and save.

# Import ArcPy and os modules
import arcpy
import os

# Set the current workspace to your Exercise11 folder
arcpy.env.workspace = r"C:\Student\PYTH\Exercise11"

# Set up variables for output path and PDF file name
outDir = r"C:\Student\PYTH\Exercise11"
finalpdf_filename = outDir + r"\FinalMapBook.pdf"

# Remove existing multi-page PDF if it exists
if os.path.exists(outDir + r"\MapPages.pdf"):
    os.remove(outDir + r"\MapPages.pdf")

# Check whether the final map book PDF exists. If it does, delete it.
if os.path.exists(finalpdf_filename):
    os.remove(finalpdf_filename)

# Create map book PDF
finalPDF = arcpy.mapping.PDFDocumentCreate(finalpdf_filename)

# Start appending pages.  Title page first.
finalPDF.appendPages(outDir + r"\PlainsViewTitlePage.pdf")

# Create MapDocument object pointing to specified mxd
mxd = arcpy.mapping.MapDocument(outDir + r"\PlainsView.mxd")

# Export Data Driven Page MXD to multi-page PDF
ddp = mxd.dataDrivenPages
print "Exporting map pages to PDF"
ddp.exportToPDF(outDir + r"\MapPages.pdf")
```

```
# Append multi-page PDF to finalPDF
finalPDF.appendPages(outDir + r"\MapPages.pdf")

# Append Contact page to PDF
finalPDF.appendPages(outDir + r"\PlainsViewContactPage.pdf")

# Set up properties for Adobe Reader and save PDF.
finalPDF.updateDocProperties(pdf_open_view = "USE_THUMBS",
                             pdf_layout = "SINGLE_PAGE")
finalPDF.saveAndClose()

# Done.  Clean up and let user know the process has finished.
del mxd, finalPDF
print "Creation of map book complete"
```

Esri data license agreement

IMPORTANT — READ CAREFULLY BEFORE OPENING THE SEALED MEDIA PACKAGE

ENVIRONMENTAL SYSTEMS RESEARCH INSTITUTE, INC. (ESRI), IS WILLING TO LICENSE THE ENCLOSED ELECTRONIC VERSION OF THIS TRAINING COURSE TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS AND CONDITIONS CONTAINED IN THIS ESRI DATA LICENSE AGREEMENT. PLEASE READ THE TERMS AND CONDITIONS CAREFULLY BEFORE OPENING THE SEALED MEDIA PACKAGE. BY OPENING THE SEALED MEDIA PACKAGE, YOU ARE INDICATING YOUR ACCEPTANCE OF THE ESRI DATA LICENSE AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS AS STATED, THEN ESRI IS UNWILLING TO LICENSE THE TRAINING COURSE TO YOU. IN SUCH EVENT, YOU SHOULD RETURN THE MEDIA PACKAGE WITH THE SEAL UNBROKEN AND ALL OTHER COMPONENTS (E.G., THE CD-ROM, TRAINING COURSE MATERIALS, TRAINING DATABASE, AS APPLICABLE) TO ESRI OR ITS AUTHORIZED INSTRUCTOR FOR A REFUND. NO REFUND WILL BE GIVEN IF THE MEDIA PACKAGE SEAL IS BROKEN OR THERE ARE ANY MISSING COMPONENTS.

ESRI DATA LICENSE AGREEMENT

This is a license agreement, and not an agreement for sale, between you (Licensee) and Esri. This Esri data license agreement (Agreement) gives Licensee certain limited rights to use the electronic version of the training course materials, training database, software, and related materials (hereinafter collectively referred to as the "Training Course"). All rights not specifically granted in this Agreement are reserved to Esri and its licensor(s).

Reservation of Ownership and Grant of License: Esri and its licensor(s) retain exclusive rights, title, and ownership to the copy of the Training Course licensed under this Agreement and hereby grant to Licensee a personal, nonexclusive, nontransferable license to use the Training Course as a single package for Licensee's own personal use only pursuant to the terms and conditions of this Agreement. Licensee agrees to use reasonable efforts to protect the Training Course from unauthorized use, reproduction, distribution, or publication.

Proprietary Rights and Copyright: Licensee acknowledges that the Training Course is proprietary and confidential property of Esri and its licensor(s) and is protected by United States copyright laws and applicable international copyright treaties and/or conventions.

Permitted Uses:

- Licensee may run the setup and install one (1) copy of the Training Course onto a permanent electronic storage device and reproduce one (1) copy of the Training Course and/or any online documentation in hard-copy format for Licensee's own personal use only.
- Licensee may use one (1) copy of the Training Course on a single processing unit.
- Licensee may make only one (1) copy of the original Training Course for archival purposes during the term of this Agreement, unless the right to make additional copies is granted to Licensee in writing by Esri.
- Licensee may use the Training Course provided by Esri for the stated purpose of Licensee's own personal GIS training and education.

Uses Not Permitted:

- Licensee shall not sell, rent, lease, sublicense, lend, assign, time-share, or transfer, in whole or in part, or provide unlicensed third parties access to the Training Course, any updates, or Licensee's rights under this Agreement.
- Licensee shall not separate the component parts of the Training Course for use on more than one (1) computer, used in conjunction with any other software package, and/or merged and compiled into a separate database(s) for other analytical uses.
- Licensee shall not reverse engineer, decompile, or disassemble the Training Course, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this restriction.
- Licensee shall not make any attempt to circumvent the technological measure(s) (e.g., software or hardware key) that effectively controls access to the Training Course, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this restriction.
- Licensee shall not remove or obscure any copyright, trademark, and/or proprietary rights notices of Esri or its licensor(s).

Term: The license granted by this Agreement shall commence upon Licensee's receipt of the Training Course and shall continue until such time that (1) Licensee elects to discontinue use of the Training Course and terminates this Agreement or (2) Esri terminates for Licensee's material breach of this Agreement. The Agreement shall automatically terminate without notice if Licensee fails to comply with any provision of this Agreement. Upon termination of this Agreement in either instance, Licensee shall return to Esri or destroy all copies of the Training Course, and any whole or partial copies, in any form and deliver evidence of such destruction to Esri, which evidence shall be in a form acceptable to Esri in its sole discretion. The parties hereby agree that all provisions that operate to protect the rights of Esri and its licensor(s) shall remain in force should breach occur.

No Implied Waivers: No failure or delay by Esri or its licensor(s) in enforcing any right or remedy under this Agreement shall be construed as a waiver of any future or other exercise of such right or remedy by Esri or its licensor(s).

Order for Precedence: This Agreement shall take precedence over the terms and conditions of any purchase order or other document, except as required by law or regulation.

Export Regulation: Licensee acknowledges that the Training Course and all underlying information or technology may not be exported or re-exported into any country to which the U.S. has embargoed goods, or to anyone on the U.S. Treasury Department's list of Specially Designated Nationals, or to the U.S. Commerce Department's Table of Deny Orders. Licensee shall not export the Training Course or any underlying information or technology to any facility in violation of these or other applicable laws and regulations. Licensee represents and warrants that it is not a national or resident of, or located in or under the control of, any country subject to such U.S. export controls.

Severability: If any provision(s) of this Agreement shall be held to be invalid, illegal, or unenforceable by a court or other tribunal of competent jurisdiction, the validity, legality, and enforceability of the remaining provisions shall not in any way be affected or impaired thereby.

Governing Law: This Agreement, entered into in the County of San Bernardino, shall be construed and enforced in accordance with and be governed by the laws of the United States of America and the State of California without reference to conflict of laws principles.

Entire Agreement: The parties agree that this Agreement constitutes the sole and entire agreement of the parties as to the matter set forth herein and supersedes any previous agreements, understandings, and arrangements between the parties relating hereto.